University of British Columbia, Vancouver

Department of Computer Science

CPSC 304 Project Cover Page

Milestone #: 2
Date: June 12, 2023
Group Number: 2

Name	Student Number	CS Alias	Preferred Email Address
Beth Koschel	28150423	g9n3c@ugrad.cs.ubc.ca	bethanylkoschel@gmail.com
Harbir Bajwa	20972261	p8i2e@ugrad.cs.ubc.ca	harbirbajwa@hotmail.com
Apram Ahuja	14367403	v0w2d@ugrad.cs.ubc.ca	apram235ahuja@gmail.com

By typing our names and student numbers in the above table, we certify that the work in the attached assignment was performed solely by those whose names and student IDs are included above. (In the case of Project Milestone 0, the main purpose of this page is for you to let us know your e-mail address, and then let us assign you to a TA for your project supervisor.)
In addition, we indicate that we are fully aware of the rules and consequences of plagiarism, as set forth by the Department of Computer Science and the University of British Columbia

2. A brief (~2-3 sentences) summary of your project. Many of your TAs are managing multiple projects so this will help them remember details about your project.

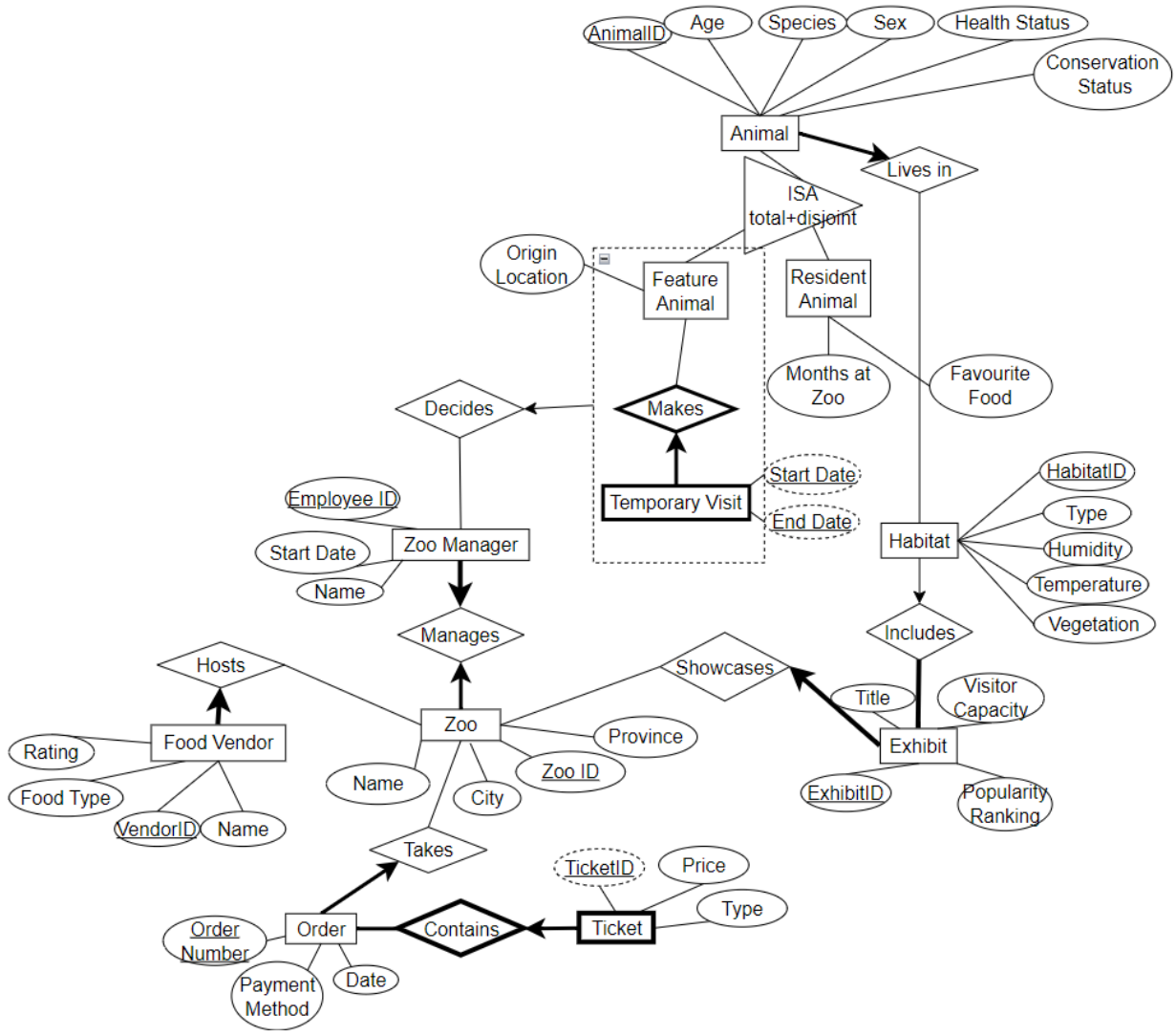
The project involves developing a Zoo Animal Management System that focuses on the relationships between various entities. The system will capture and model the interactions between zoos, zoo managers, orders, animals, exhibits, habitats, and food vendors. It will provide functionality to manage exhibits, track animal visits, analyze visitor demographics, monitor revenue from ticket sales. The database will enable efficient zoo operations and aid decision-making processes related to exhibit popularity, animal selection, and vendor management.

3. The ER diagram you are basing your item #3 (below) on. This ER diagram may be the same as your milestone 1 submission or it might be different. If you have made changes from the version submitted in milestone 1, attach a note indicating what changes have been made and why.
If you have decided not to implement the suggestions given by your project mentor, please be sure to leave a note stating why. This is not to say that you must do everything that your project mentor says. In many instances, there are trade-offs between design choices and your decision may be influenced by different factors. That being said, your TAs will often leave suggestions that are meant to help massage your project into a form that will fit with the requirements in future project milestones. If you choose not to take their advice, it would be helpful for them to know why in order to better assist the group moving forward.

We did not make changes regarding the feedback given “In order to find the most popular exhibit / animal, you probably would want to associate the animal / exhibit information somehow in the order”. It does not make sense to have animal popularity because animals are individual in our ERD. tigerA and tigerB live in the same habitat so tigerA cannot be more popular than tigerB. However, the tiger habitat can have an associated popularity (covers all tigers). And we already have an association between habitats and exhibits.

Some of the other changes made to ERD are as follows:

- Exhibits have total participation with Showcases. (All exhibits must be associated with a zoo to exist). So that Exhibits will have zooID as a foreign key.
- Changed Animal.name to “species”. So instead of each animal having their own name such as “Alex”, they would have an attribute of their species. Lions would be “Panthera leo”. This will add a functional dependency between species and conservation status.



4.The schema derived from your ER diagram (above). For the translation of the ER diagram to the relational model, follow the same instructions as in your lectures. The process should be reasonably straightforward. For each table:

a. List the table definition (e.g., Table1(attr1: domain1, attr2: domain2, ...)). Make sure to include the domains for each attribute.
b. Specify the primary key (PK), candidate key, (CK) foreign keys (FK), and other constraints (e.g., not null, unique, etc.) that the table must maintain.

a.

- Zoo (zooID: INTEGER, city: CHAR(32), zName: CHAR(32), province: CHAR(4), **EmployeeID**: INTEGER)

- ZooManager (eName: CHAR(32), employeeID: INTEGER, joinDate: DATE, **zooID**: INTEGER)
- Order (orderNumber: INTEGER, orderDate: DATE, paymentMethod: CHAR(4), **zooID**: INTEGER)
- Ticket (ticketID: INTEGER, **orderNumber**: INTEGER, ticketType: CHAR(4), price:REAL)
- FeatureAnimal (animalID: INTEGER, **habitatID**: INTEGER, species: CHAR(32), sex: CHAR(2), age: INTEGER, healthStatus: CHAR(4), conservationStatus: CHAR(4), originLocation: CHAR(32))
- ResidentAnimal (animalID: INTEGER, **habitatID**: INTEGER, species: CHAR(32), sex: CHAR(2), age: INTEGER, healthStatus: CHAR(4), conservationStatus: CHAR(4), monthsAtZoo: INTEGER, favouriteFood: CHAR(32))
- TemporaryVisit (startDate: DATE, endDate: DATE, **animalID**: INTEGER, **EmployeeID**: INTEGER)
    - (Combined TemporaryVisit, Makes, and the whole aggregation into one table)
- Exhibits (exhibitID: INTEGER, exhibitTitle: CHAR(32), visitorCapacity: INTEGER, popularityRating: INTEGER, **zooID**: INTEGER)
- Habitat (habitatID: INTEGER, **exhibitID**: INTEGER, habitatType: CHAR(32), humidity: REAL, temperature: REAL, vegetation: CHAR(32))
- FoodVendor (vendorID: INTEGER, vName: CHAR(32), rating: REAL, foodType: CHAR(32), **zooID**: INTEGER)

b.

Zoo
Primary key: zooID
Candidate key: zooID
Foreign key: employeeID NOT NULL, UNIQUE

ZooManager
Primary key: employeeID
Candidate key: employeeID
Foreign key: zooID NOT NULL, UNIQUE

Order
Primary key: orderNumber
Candidate key: orderNumber
Foreign key: zooID NOT NULL

Ticket

Primary key: ticketID, orderNumber
Candidate key: ticketID, orderNumber
Foreign key: orderNumber

FeatureAnimal
Primary key: animalID
Candidate key: animalID
Foreign key: habitatID NOT NULL

ResidentAnimal
Primary key: animalID
Candidate key: animalID
Foreign key: habitatID NOT NULL

TemporaryVisit
Primary key: {animalID, startDate, endDate}
Candidate key: {animalID, startDate, endDate}
Foreign key: animalID, employeeID

Exhibits
Primary key: exhibitID
Candidate key: exhibitID
Foreign key: zooID NOT NULL

Habitat
Primary key: habitatID
Candidate key: habitatID
Foreign key: exhibitID

FoodVendor
Primary key: vendorID
Candidate key: vendorID
Foreign key: zooID NOT NULL

5. Functional Dependencies (FDs)
   a. Identify the functional dependencies in your relations, including the ones involving all candidate keys (including the primary key).

      PKs and CKs are considered functional dependencies and should be included in the list of FDs. You do not need to include trivial FDs such as A → A.

Note: In your list of FDs, there must be some kind of valid FD other those identified by a PK or CK. If you observe that no relations have FDs other than the PK and CK(s), then you will have to intentionally add some (meaningful) attributes to show valid FDs. We want you to get a good normalization exercise. Your design must go through a normalization process.

PKs & CKs FD :

ZooID -> City, Province, zName, EmployeeID
EmployeeID -> eName, joinDate, ZooID
VendorID -> vName, FoodType, Rating, ZooID
ExhibitID -> Title, VisitorCapacity, PopularityRanking, ZooID
HabitatID -> habitatType, Humidity, Temperature, Vegetation
AnimalID -> Age, species, Sex, HealthStatus, ConservationStatus, HabitatID, originLocation
AnimalID -> Age, species, Sex, HealthStatus, ConservationStatus, HabitatID, monthsAtZoo, favouriteFood
OrderNumber ->Payment Method, Date, ZooID
OrderNumber, TicketID -> Price, Type
AnimalID, StartDate, EndDate -> OriginLocation, EmployeeID

Other valid FDs :

Species -> Conservation status
    example; panda -> endangered
habitatType -> humidity, temperature
    example; Rainforest -> 100%, warm
ticketType -> price
    example; General Admission -> $15
zName -> city
    Assuming zoos are named after their city.Example; Vancouver Zoo -> Vancouver
vName -> foodType
    example; Japadog -> Hotdogs

6. Normalization

a. Normalize each of your tables to be in 3NF or BCNF. Give the list of tables, their primary keys, their candidate keys, and their foreign keys after normalization.

   You should show the steps taken for the decomposition. Should there be errors, and no work is shown, no partial credit can be awarded without steps shown. The format should be the same as Step 3, with tables listed similar to Table1(attr1:domain1, attr2:domain2, …). ALL Tables must be listed, not only the ones post normalization.

To determine if our relations were in BCNF we used the lossless-join BCNF decomposition algorithm. The majority of our relations were in BCNF already because they were PK/CK dependencies:

Relation: **Zoo** (zooID, city, zName, province, **EmployeeID**)

- **FD**: zooID -> City, Province, zName, EmployeeID
    - This satisfies BCNF because zooID is already the key
- **FD**: zName -> city
    - Violates BCNF because zName is not a superkey
- Decompose to **Zoo1**(zName, city) and **Zoo2**(zooID, zName, province, EmployeeID)
    - Both FDs satisfy BCNF

Relation: featureAnimal(animalID, **habitatID**, species, sex, age, healthStatus, conservationStatus, originLocation)

- **FD**: AnimalID -> Age, species, Sex, HealthStatus, ConservationStatus, HabitatID, originLocation
- **FD**: species -> ConservationStatus
    - Violates BCNF because species is not a superkey of featureAnimal. Decompose

1. **animal1**(species, ConservationStatus)
    - Animal1 is in BCNF (species is the key)
2. **featureAnimal1**(animalID, habitatID, species, sex, age, healthStatus, originLocation)
    - featureAnimal1 is in BCNF (animalID is the key)

Relation: residentAnimal(animalID, **habitatID**, species, sex, age, healthStatus, conservationStatus, monthsAtZoo, favouriteFood)

- **FD**: AnimalID -> Age, species, Sex, HealthStatus, ConservationStatus, HabitatID, monthsAtZoo, favouriteFood
- **FD**: species -> ConservationStatus
    - Violates BCNF because species is not a superkey of residentAnimal
- Decomposed into two relations.

1. The **relation**(species, ConservationStatus) can be combined with the relation **animal1(**species, ConservationStatus) from decomposing featureAnimal relation above.
2. **residentAnimal1**(animalID, habitatID, species, sex, age, healthStatus, monthsAtZoo, favouriteFood)
    - animal1 is in BCNF (species is the key)
    - residentAnimal1 is in BCNF (animalID is the key)

Relation: TemporaryVisit(animalID, startDate, endDate , employeeID)

- **FD**: animalID,startDate,endDate -> employeeID
    - This satisfies BCNF because {animalID, startDate, endDate} is already the key.

Relation: ZooManager(employeeID, joinDate, eName, **zooID**)

- **FD**: employeeID -> joinDate,eName,zooID
    - This satisfies BCNF because employeeID is the key

Relation: FoodVendor(vendorID, vName, foodType, rating, **zooID**)

- **FD** : vendorID -> vName, foodType, rating, zooID
    - This satisfies BCNF because vendorID is the key
- **FD** : vName -> foodType
    - Violates BCNF as vName is not a superkey in FoodVendor Relation i.e., we decompose this into two relations:
        1. **FoodVendor1**(vName, foodType)
            - This adheres to BCNF as vName is a key in FoodVendor1
        2. **FoodVendor2**(vendorID, vName, rating, zooID)
            - This adheres to BCNF as vendorID-> vName,rating,zooID i.e., vendorID is already a key in FoodVendor2 relation.

Relation: Habitat(habitatID, **exhibitID**, habitatType, humidity, temperature, vegetation)

- **FD**: HabitatID -> habitatType, Humidity, Temperature, Vegetation
- **FD**: habitatType -> humidity, temperature
    - Violates BCNF as habitatType is not a superkey.
- Decomposed into two relations:
    1. **Habitat1** (habitatType, humidity, temperature).
        - Satisfies BCNF as habitatType is a superkey of Habitat1
    2. **Habitat2** (habitatID, exhibitID, habitatType, vegetation)
        - Satisfies BCNF as habitatID is a superkey of Habitat2

Relation: **Order**(orderNumber , orderDate, paymentMethod, zooID)

- **FD** : orderNumber -> orderDate, paymentMethod,zooID
    - This is in BCNF as orderNumber is already a key.

Relation : Ticket(orderNumber, ticketID , price , ticketType)

- **FD**: orderNumber, ticketID -> price, ticketType
    - This is in BCNF as orderNumber and ticketID is already a key which gives price for the ticketType at a particular zoo.
- **FD**: ticketType -> price
    - This doesn't adhere to BCNF as ticketType is not a superkey in Ticket relation i.e., we decompose it into two relations :
        1. **Ticket1**(ticketType, price)
            - This adheres to BCNF as ticketType is a superkey in Ticket1
        2. **Ticket2**(orderNumber, ticketID , ticketType)

    - This adheres to BCNF as orderNumber and ticketID are a superkey in Ticket2

Relation: **Exhibits**(exhibitID , title, visitorCapacity, popularityRanting, zooID)

- FD : exhibitID -> title, visitorCapacity, popularityRating, zooID
    - This adheres to BCNF as exhibitID is already a key in exhibit relation.

Since all of our relations are now in BCNF we created the following tables with their primary keys, candidate keys and foreign keys:

- Zoo1 (zName: CHAR(32) city: CHAR(32))
- Zoo2 (zooID: INTEGER, zName: CHAR(32), province: CHAR(4), **EmployeeID**: INTEGER)
- ZooManager (eName: CHAR(32), employeeID: INTEGER, joinDate: DATE, **zooID**: INTEGER)
- Order (orderNumber: INTEGER, orderDate: DATE, paymentMethod: CHAR(4), **zooID**: INTEGER)
- Ticket1 (ticketType: CHAR(4), price:REAL)
- Ticket2 (ticketID: INTEGER, **orderNumber**: INTEGER, ticketType: CHAR(4))
- Animal1 (species: CHAR(32), conservationStatus: CHAR(4))
- FeatureAnimal1 (animalID: INTEGER, **habitatID**: INTEGER, species: CHAR(32), sex: CHAR(2), age: INTEGER, healthStatus: CHAR(4), originLocation: CHAR(32))
- ResidentAnimal1 (animalID: INTEGER, **habitatID**: INTEGER, species: CHAR(32), sex: CHAR(2), age: INTEGER, healthStatus: CHAR(4), monthsAtZoo: INTEGER, favouriteFood: CHAR(32))
- TemporaryVisit (startDate: DATE, endDate: DATE, **animalID**: INTEGER, **EmployeeID**: INTEGER)
    - (Combined TemporaryVisit, Makes, and the whole aggregation into one table)
- Exhibits (exhibitID: INTEGER, exhibitTitle: CHAR(32), visitorCapacity: INTEGER, popularityRating: INTEGER, **zooID**: INTEGER)
- Habitat1 (habitatType: CHAR(32), humidity: REAL, temperature: REAL)
- Habitat2 (habitatID: INTEGER, **exhibitID**: INTEGER, habitatType: CHAR(32), vegetation: CHAR(32))
- FoodVendor1 (vName: CHAR(32), foodType: CHAR(32))
- FoodVendor2 (vendorID: INTEGER, vName: CHAR(32), rating: REAL, **zooID**: INTEGER)

7. The SQL DDL statements required to create all the tables from item #6. The statements should use the appropriate foreign keys, primary keys, UNIQUE constraints, etc.

```
CREATE TABLE Zoo1 (
   zName       CHAR(32) PRIMARY KEY,
   city        CHAR(32)
)

CREATE TABLE Zoo2 (
  zooID        INTEGER,
  zName        CHAR(32),
  province     CHAR(32),
  employeeID   INTEGER NOT NULL,
  UNIQUE employeeID,
  PRIMARY KEY (zooID),
  FOREIGN KEY (employeeID)
        REFERENCES ZooManager(employeeID)
                ON DELETE CASCADE
                        ON UPDATE CASCADE

)

CREATE TABLE ZooManager (
   eName       CHAR(32),
   employeeID  INTEGER,
   joinDate    DATE,
   zooID       INTEGER NOT NULL,
   UNIQUE zooID,
   PRIMARY KEY (employeeID),
   FOREIGN KEY (zooID)
      REFERENCES Zoo(zooID)
         ON DELETE CASCADE
                ON UPDATE CASCADE
)

CREATE TABLE Order (
   orderNumber     INTEGER,
   orderDate       DATE,
   paymentMethod   CHAR(4),
   zooID           INTEGER NOT NULL,
   PRIMARY KEY (orderNumber),
   FOREIGN KEY (zooID)
```

```
        REFERENCES Zoo(zooID)
            ON DELETE CASCADE
                    ON UPDATE CASCADE
)


CREATE TABLE Ticket1 (
    ticketType      CHAR(4) PRIMARY KEY,
    price           REAL,

)

CREATE TABLE Ticket2 (
    ticketID        INTEGER,
    orderNumber     INTEGER,
    ticketType    CHAR(4),
    PRIMARY KEY (ticketID, orderNumber),
    FOREIGN KEY (orderNumber)
        REFERENCES Order(orderNumber)
            ON DELETE CASCADE
                    ON UPDATE CASCADE
)

CREATE TABLE Animal1 (
    species             CHAR(32) PRIMARY KEY,
    conservationStatus  CHAR(4)
)


CREATE TABLE ResidentAnimal1 (
    animalID            INTEGER,
    habitatID           INTEGER NOT NULL,
    species             CHAR(32)
    sex                 CHAR(2),
    age                 INTEGER,
    healthStatus        CHAR(4),
    conservationStatus  CHAR(4),
    monthsAtZoo         INTEGER,
    favouriteFood       CHAR(32),
    PRIMARY KEY (animalID),
    FOREIGN KEY (habitatID)
        REFERENCES Habitat(habitatID)
            ON DELETE CASCADE
                    ON UPDATE CASCADE
```

```
)

CREATE TABLE FeatureAnimal1 (
   animalID           INTEGER,
   habitatID          INTEGER NOT NULL,
   species            CHAR(32),
   sex                CHAR(2),
   age                INTEGER,
   healthStatus       CHAR(4),
   originLocation     CHAR(32),
   PRIMARY KEY (animalID),
   FOREIGN KEY (habitatID)
      REFERENCES Habitat(habitatID)
         ON DELETE CASCADE
                  ON UPDATE CASCADE
)

CREATE TABLE TemporaryVisit (
   startDate       DATE,
   endDate         DATE,
   animalID        INTEGER,
   employeeID      INTEGER,
   PRIMARY KEY (startDate, endDate, animalID),
   FOREIGN KEY (employeeID)
      REFERENCES ZooManager(employeeID)
         ON DELETE CASCADE
  FOREIGN KEY (animalID)
      REFERENCES FeatureAnimal1(animalID)
         ON DELETE CASCADE
                  ON UPDATE CASCADE
)

CREATE TABLE Exhibits (
   exhibitID          INTEGER,
   exhibitTitle       CHAR(32),
   visitorCapacity    INTEGER,
   popularityRating   INTEGER,
   zooID              INTEGER NOT NULL,
   PRIMARY KEY (exhibitID),
   FOREIGN KEY (zooID)
      REFERENCES Zoo(zooID)
         On DELETE CASCADE
                  ON UPDATE CASCADE
)
```

```
CREATE TABLE Habitat1 (
   habitatType     CHAR(32),
   humidity        REAL,
   temperature     REAL,
   PRIMARY KEY (habitat type)
)

CREATE TABLE Habitat2 (
   habitatID       INTEGER,
   exhibitID       INTEGER,
   habitatType     CHAR(32),
   vegetation      CHAR(32),
   PRIMARY KEY (habitatID),
   FOREIGN KEY (exhibitID)
      REFERENCES Exhibits(exhibitID)
         ON DELETE CASCADE
                  ON UPDATE CASCADE
)

CREATE TABLE FoodVendor1 (
   vName           CHAR(32) PRIMARY KEY,
   foodType        CHAR(32)
)

CREATE TABLE FoodVendor2 (
   vendorID        INTEGER,
   vName           CHAR(32),
   rating          REAL,
   zooID           INTEGER NOT NULL,
   PRIMARY KEY (vendorID),
   FOREIGN KEY (zooID),
      REFERENCES Zoo(zooID)
         ON DELETE CASCADE
                  ON UPDATE CASCADE
)
```

8. INSERT statements to populate each table with at least 5 tuples. You will likely want to have more than 5 tuples so that you can have meaningful queries later on.

```
INSERT INTO Zoo1 (zName: CHAR(32) city: CHAR(32))
VALUES
 ('Toronto Zoo', 'Toronto'), ('Greater Vancouver Zoo', 'Vancouver'), ('Winnipeg Zoo', 'Winnipeg'),
 ('Victoria Bug Zoo', 'Victoria'), ('Edmonton Valley Zoo', 'Edmonton');
```

```
INSERT INTO Zoo2 (zooID, zName, province, EmployeeID)
VALUES
  (1, 'Toronto Zoo', 'ON', 101),
  (2, 'Greater Vancouver Zoo', 'BC', 102),
  (3, 'Winnipeg Zoo', 'MB', 103),
  (4, 'Victoria Bug Zoo', 'BC', 104),
  (5, 'Edmonton Valley Zoo', 'AB', 105),
  (6, 'Montreal Biodome', 'QC, 106);

INSERT INTO ZooManager (eName, employeeID, joinDate, zooID)
VALUES
  ('John Smith', 101, '2022-01-15', 1),
  ('Emily Johnson', 102, '2021-06-28', 2),
  ('Michael Williams', 103, '2023-03-10', 3),
  ('Sophia Brown', 104, '2022-09-05', 4),
  ('Daniel Davis', 105, '2021-12-01', 5),
  ('Olivia Wilson', 106, '2023-04-18', 6);

INSERT INTO Order (orderNumber, orderDate, paymentMethod, zooID)
VALUES
  (101, '2023-01-15', 'CC', 1),
  (102, '2023-02-20', 'PayPal', 2),
  (103, '2023-03-10', 'CC', 1),
  (104, '2023-04-05', 'Cash', 3),
  (105, '2023-05-18', 'PayPal', 2),
  (106, '2023-06-01', 'CC', 4);

INSERT INTO Ticket1 (ticketType, price)
VALUES
  ('Adult', 20.99),
  ('Child', 10.99),
  ('Senior', 15.99),
  ('Student', 17.99),
  ('Family', 45.99),
  ('Group', 12.99);

INSERT INTO Ticket2 (ticketID, orderNumber, ticketType)
VALUES
  (1001, 101, 'Adult'),
  (1002, 102, 'Child'),
  (1003, 103, 'Adult'),
  (1004, 104, 'Senior'),
  (1005, 105, 'Child'),
```

```
  (1006, 106, 'Student');

INSERT INTO Animal1 (species, conservationStatus)
VALUES
  ('Acinonyx jubatus', 'LC'),
  ('Panthera pardus', 'NT'),
  ('Antilocapra americana', 'LC'),
  ('Lamprotornis superbus', 'LC'),
  ('Ateles belzebuth', 'VU'),
  ('Gazella dorcas', 'LC'),
  ('Giraffa camelopardalis', 'VU'),
  ('Panthera tigris', 'EN'),
  ('Bison bison', 'LC'),
  ('Ploceus cucullatus', 'LC'),
  ('Alouatta caraya', 'LC'),
  ('Oryx gazella', 'LC');

INSERT INTO FeatureAnimal1 (animalID, habitatID, species, sex, age, healthStatus,
originLocation)
VALUES
  (13, 1, 'Acinonyx jubatus', 'M', 5, 'Good', 'Africa'),
  (14, 2, 'Panthera pardus', 'F', 7, 'Excellent', 'Asia'),
  (15, 3, 'Antilocapra americana', 'M', 3, 'Good', 'North America'),
  (16, 4, 'Lamprotornis superbus', 'F', 4, 'Good', 'Africa'),
  (17, 5, 'Ateles belzebuth', 'M', 6, 'Fair', 'South America'),
  (18, 6, 'Gazella dorcas', 'F', 2, 'Good', 'Africa');

INSERT INTO ResidentAnimal1 (animalID, habitatID, species, sex, age, healthStatus,
monthsAtZoo, favouriteFood)
VALUES
  (19, 1, 'Giraffa camelopardalis', 'F', 9, 'Good', 18, 'Leaves'),
  (20, 2, 'Panthera tigris', 'M', 6, 'Excellent', 12, 'Meat'),
  (21, 3, 'Bison bison', 'M', 4, 'Good', 24, 'Grass'),
  (22, 4, 'Ploceus cucullatus', 'F', 3, 'Good', 9, 'Seeds'),
  (23, 5, 'Alouatta caraya', 'M', 5, 'Fair', 15, 'Fruits'),
  (24, 6, 'Oryx gazella', 'F', 7, 'Good', 6, 'Grass');

INSERT INTO TemporaryVisit (startDate, endDate, animalID, EmployeeID)
VALUES
  ('2023-01-15', '2023-01-20', 13, 101),
  ('2023-02-05', '2023-02-12', 14, 102),
  ('2023-03-10', '2023-03-15', 15, 103),
  ('2023-04-01', '2023-04-07', 16, 104),
  ('2023-05-18', '2023-05-25', 17, 105),
```

```
  ('2023-06-10', '2023-06-17', 18, 106);

INSERT INTO Exhibit1 (exhibitID, exhibitTitle, visitorCapacity, popularityRating, zooID)
VALUES
  (1, 'Big Cat', 100, 9, 1),
  (2, 'Rainforest Adventure', 150, 8, 2),
  (3, 'Grassland Safari', 80, 7, 1),
  (4, 'Desert Oasis', 50, 6, 3),
  (5, 'Tropical Paradise', 120, 9, 2),
  (6, 'Savanna Safari', 90, 7, 1);

INSERT INTO Habitat1 (habitatType, humidity, temperature)
VALUES
  ('Savanna', 60.5, 30.2),
  ('Rainforest', 80.2, 25.8),
  ('Desert', 20.8, 40.6),
  ('Grassland', 55.1, 28.5),
  ('Arctic', 30.7, -10.2),
  ('Woodland', 65.3, 24.9);

INSERT INTO Habitat2 (habitatID, exhibitID, habitatType, vegetation)
VALUES
  (1, 1, 'Savanna', 'Grass'),
  (2, 2, 'Rainforest', 'Trees'),
  (3, 3, 'Grassland', 'Grass'),
  (4, 4, 'Desert', 'Cacti'),
  (5, 2, 'Rainforest', 'Vines'),
  (6, 1, 'Savanna', 'Trees');


 INSERT INTO FoodVendor1 (vName, foodType)
VALUES
  ('Japadog', 'Hot Dogs'),
  (‘Rain or Shine', ‘Ice-Cream’),
  ('Sushi Spot', 'Japanese'),
  ('Taco Truck', 'Mexican'),
  ('Pizza Palace', 'Italian'),
  ('Crepes Corner', 'French');

INSERT INTO FoodVendor2 (vendorID, vName, rating, zooID)
VALUES
  (1, 'Japadog', 4.2, 1),
  (2, ‘Rain or Shine’, 4.5, 2),
  (3, 'Sushi Spot', 4.1, 1),
```

```
(4, 'Taco Truck', 4.3, 3),
(5, 'Pizza Palace', 4.4, 2),
(6, 'Crepes Corner', 4.0, 1);
```