CPSC 304 Project Cover Page

Milestone #: ___1___

Date: __02/27/2025___

Group Number: ____60_____

Name	Student Number	CS Alias (Userid)	Preferred E-mail Address
Mukund Patil	97159511	mpatil01	mukunda05@gmail.com
Irwin Wang	91618181	iwang08	irwinwanguni@gmail.com
Aadetri Tawara	61166476	c5s1h	aadetritawara@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

Summary

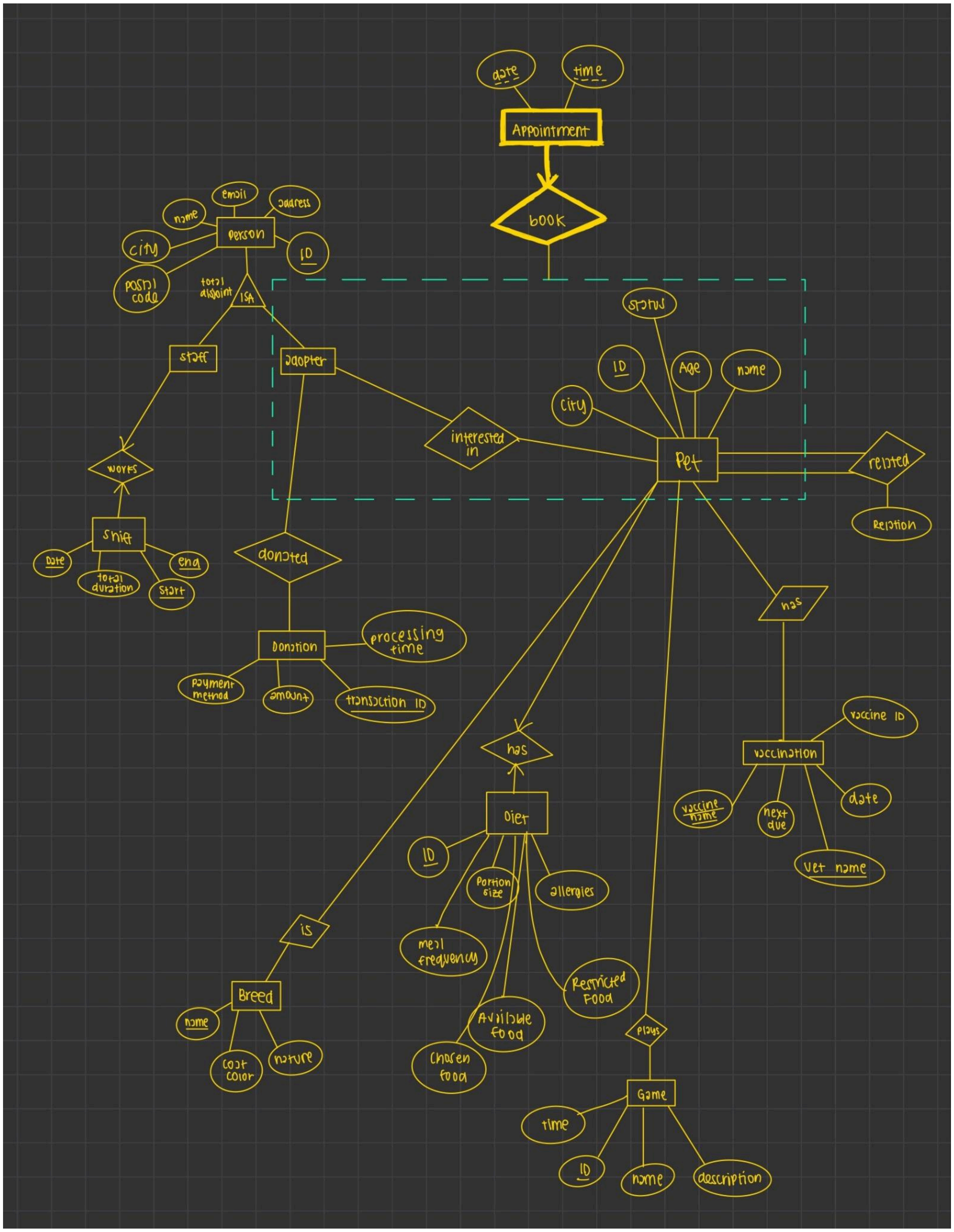
Our project, Find My Pet, aims to assist potential candidates interested in adopting a pet by modelling key aspects of a successful pet adoption agency. It will store valuable information about pets like diets, genetics, nature, vaccinations and games they like to play, and match pets with suitable adopters by setting up appointments.

Changes

Changes made since milestone 1:

1. Made weak entity into a one-to-many relationship. Now, one pet and adopter pair can book multiple appointments. This change was made because a weak entity must be in a one-to-many relationship, and ours was a one-to-one relationship before.
2. Changed the ISA relationship so that the parent entity (Person) contains the primary key rather than the subentities. The key for the parent and sub entities is now "ID". This change was made because the sub entities inherit the key from its super entity, and previously, our parent entity didn't have any key.
3. Added a "relation" attribute to the Related Relationship. This was added because it makes sense to include information about how two pets are related (e.g. parent, sibling, etc.).
4. Added two new attributes to Person (city and postal code). This allows us to log and keep track of additional information.
5. We chose not to implement phone numbers as a key because phone numbers could change and it provided ambiguity regarding keys and ID's, so we instead added ID to person, and adopter and staff both inherit an ID, as Adopter ID and Staff ID respectively.

ER Diagram


date
time
Appointment
book
email
address
name
person
city
ID
postal code
total disjoint
ISA
staff
adopter
works
shift
Date
total duration
end
start
donated
Donation
processing time
payment method
amount
transaction ID
interested in
status
ID
Age
name
city
Pet
related
Relation
has
vaccination
vaccine ID
vaccine name
next due
date
vet name
has
Diet
ID
portion size
allergies
meal frequency
Restricted Food
Available food
chosen food
is
Breed
name
coat color
nature
plays
Game
time
ID
name
description

<u>Relational Schema</u>

<u>Staff</u>

Staff (

Name : VARCHAR(50),

Email : VARCHAR(255),

Address : VARCHAR(255),

<u>Staff ID</u> : INTEGER,

Postal Code : VARCHAR(6),

City : VARCHAR(20))

PRIMARY KEY(Staff ID),   <- PK

UNIQUE(Email)   <- CK

<u>Adopter</u>

Adopter(

Name : VARCHAR(50),

Email : VARCHAR(255),

Address : VARCHAR(255),

<u>Adopter ID</u> : INTEGER,

Postal Code : VARCHAR(6),

City : VARCHAR(20))

PRIMARY KEY(Adopter ID)   <- PK

UNIQUE(Email)   <- CK

<u>**Works_Shift**</u>

Works_Shift(

<u>Date</u> : DATE,

Total Duration : INTEGER,

<u>Start</u> : TIME,

<u>End</u> : TIME,

Staff ID: VARCHAR(10),

Shift Type: VARCHAR(20))

PRIMARY KEY(Date, Start, End)   <- PK

FOREIGN KEY (Staff ID) REFERENCES Staff   <- FK

FOREIGN KEY (Shift Type) REFERENCES Shift   <- FK

<u>**Donation**</u>

Donation(

Payment method : CHAR(4),

Amount : DECIMAL(10, 2),

<u>Transaction ID</u>: INTEGER,

Adopter ID: INTEGER)

PRIMARY KEY(Transaction ID)   <- PK

FOREIGN KEY(Adopter ID)   <- FK

Donated

Donated(**Transaction ID** : INTEGER,

Adopter ID : INTEGER)

PRIMARY KEY(Transaction ID)   <- PK

FOREIGN KEY (AdopterID) REFERENCES Adopter   <- FK

FOREIGN KEY (Transaction ID) REFERENCES Donation   <- FK

Pet

Pet(

City : VARCHAR(20),

PetID : INTEGER,

Status : BOOLEAN,

Age : INTEGER,

Name : VARCHAR(50))

PRIMARY KEY(ID)   <- PK

<u>Interested_In</u>

Interested_In(<u>**Pet ID**</u> : INTEGER,

<u>**Adopter ID**</u>: VARCHAR(10))

PRIMARY KEY(Pet ID, Adopter ID) <- PK

FOREIGN KEY (Pet ID) REFERENCES Pet(ID) <- FK

FOREIGN KEY (Phone #) REFERENCES Adopter <- FK

<u>Related</u>

Related(<u>**Pet ID**</u> : INTEGER,

<u>**Related Pet ID**</u> : INTEGER,

Relation : VARCHAR(20))

PRIMARY KEY (Pet ID, Related Pet ID) <- PK

FOREIGN KEY (Pet ID) REFERENCES Pet(ID) <- FK

FOREIGN KEY (Related Pet ID) REFERENCES Pet(ID) <- FK

<u>Vaccination</u>

Vaccination(Vaccine Name: VARCHAR(50),

Next Due: DATE,

Vet Name: VARCHAR(50),

Date: DATE,

<u>Vaccine ID</u>: INTEGER,

Effectiveness Period: INTEGER)

PRIMARY KEY(Vaccine ID) <- PK

<u>Has_Vaccine</u>

Has_Vaccine(**<u>Vaccine ID</u>**: INTEGER,

<u>Pet ID</u>: INTEGER)

PRIMARY KEY (Vaccine ID, Pet ID) <- PK

FOREIGN KEY (Vaccine ID) REFERENCES Vaccination <- FK

FOREIGN KEY (Pet ID) REFERENCES Pet(ID) <- FK

<u>Diet</u>

Diet(<u>ID</u>: INTEGER,

Portion Size: DECIMAL(10,2),

Allergies: TEXT[],

Meal Frequency: INTEGER,

Portion Size : DECIMAL(10,2),

Allergies : TEXT[],

Calories : TEXT[],

Chosen Foods: TEXT[],

Available Foods : TEXT[],

Restricted Foods : TEXT[],

Diet ID: INTEGER)

PRIMARY KEY (Diet ID)   <- PK

Has_Diet(**Diet ID**: INTEGER,

Pet ID: INTEGER)

PRIMARY KEY (Diet ID, Pet ID)   <- PK

FOREIGN KEY (Diet ID) REFERENCES Diet(ID)   <- FK

FOREIGN KEY (Pet ID) REFERENCES Pet(ID)   <- FK

Game(Time: TIME,

ID: INTEGER,

Name: VARCHAR(50),

Description: VARCHAR(255))

PRIMARY KEY (ID)   <- PK

UNIQUE(Name)   <- CK

Plays

Plays(**Game ID:** INTEGER,

Pet ID: INTEGER)

PRIMARY KEY (Game ID, Pet ID)   <- PK

FOREIGN KEY (Game ID) REFERENCES Game(ID)   <- FK

FOREIGN KEY (Pet ID) REFERENCES Pet(ID)   <- FK

Book_Appointment

Book_Appointment(Date: DATE,

Time: TIME,

Adopter ID: VARCHAR(10) NOT NULL,

Pet ID: INTEGER   NOT NULL)

PRIMARY KEY (Date, Adopter ID, Pet ID)<- PK

FOREIGN KEY (Adopter ID) REFERENCES Adopter   <- FK

FOREIGN KEY (Pet ID) REFERENCES Pet(ID)   <- FK

Breed

Breed(Name : VARCHAR(50),

Coat Color : VARCHAR(10),

Nature : VARCHAR(20))

PRIMARY KEY(Name)   <- PK

Is_Breed

Is_Breed(**Name**: VARCHAR(50),

Pet ID : INTEGER)

PRIMARY KEY(Name, Pet ID)   <- PK

FOREIGN KEY (Name) REFERENCES Breed <- FK

FOREIGN KEY (Pet ID) REFERENCES Pet(ID) <- FK

Functional Dependencies (FDs)

1. Staff

- PK/CK FDs:
  - Staff ID -> Name, Email, Address, Postal Code, City (PK-> all attributes)
  - Email -> Staff ID (CK -> all attributes)
- Non-PK/CK FDs:
  - Address -> Postal Code, City

2. Adopter

- PK/CK FDs:
  - Adopter ID -> Name, Email, Address, Postal Code, City(PK -> all attributes)
  - Email -> Adopter ID (CK -> all attributes)
- Non-PK/CK FDs:
  - Address -> Postal Code, City

3. Works_Shift

- PK/CK FDs:
  - Date, Start, End -> Shift Type, Total Duration, Staff ID (PK -> Staff working the shift)
  - Staff ID -> Date, Start, End Shift Type, Total Duration (CK -> Staff working the shift)
- Non-PK/CK FDs:
  - Total Duration, Start -> End (for example, full-time or part time)
  - Start, End -> Shift Type

4. Donation

- PK/CK FDs:
    - Transaction ID -> Payment method, Amount, Adopter ID (PK -> all attributes)

5. Donated

- PK/CK FDs:
    - Transaction ID -> {Transaction ID, Adopter ID} (PK -> all attributes)

6. Pet

- PK/CK FDs:
    - PetID -> City, Status, Age, Name (PK -> all attributes)
- Non-PK/CK FDs:
    - City -> Status (Some pets may only be available for adoption in certain cities)

7. Interested_In

- PK/CK FDs:
    - {Pet ID, Adopter ID} -> {Pet ID, Adopter ID} (PK -> all attributes)

8. Related

- PK/CK FDs:
    - {Pet ID, Related Pet ID} -> Relation (PK -> Relationship type)

9. Vaccination

- PK/CK FDs:
    - {Vaccine Name, Vet Name} -> Next Due, Date, Vaccine ID, Effectiveness Period (CK -> all attributes)
    - Vaccine ID -> Vaccine Name, Vet Name, Next Due, Date (PK -> all

attributes)

- Non-PK/CK FDs:
    - Vaccine Name -> Effectiveness Period (Vaccines have a known duration of effectiveness)
    - {Date, Effectiveness Period} -> Next Due
    - {Date, Next Due} -> Effectiveness Period

10. Has_Vaccine

- PK/CK FDs:
    - {Vaccine Name, Pet ID} -> {Vaccine Name, Pet ID} (PK -> all attributes)

11. Diet

- PK/CK FDs:
    - Diet ID -> Portion Size, Allergies, Meal Frequency, Restricted Foods, Calories, Chosen Foods, Available Foods (PK -> all attributes)
- Non-PK/CK FDs:
    - {Portion Size, Calories} -> Meal Frequency (Larger portion sizes mean less meals)
    - Allergies -> Restricted Foods (Certain allergies require avoiding specific foods)
    - {Avilable Foods, Restricted Foods} -> Chosen Foods
    - Chosen Foods -> Calories

12. Has_Diet

- PK/CK FDs:
    - {Diet ID, Pet ID} -> {Diet ID, Pet ID} (PK -> all attributes)

13. Game

- PK/CK FDs:
    - Game ID -> Name, Description, Time (PK -> all attributes)
    - Name -> ID, Description, Time (Candidate Key -> all attributes)

14. Plays

- PK/CK FDs:
    - {Game ID, Pet ID} -> {Game ID, Pet ID} (PK -> all attributes)

15. Book_Appointment

- PK/CK FDs:
    - {Date, Adopter ID, Pet ID} -> {Date, Time, Adopter ID, Pet ID} (PK -> all attributes)

16. Breed

- PK/CK FDs:
    - Name -> Coat Color, Nature (PK -> all attributes)

17. Is_Breed

- PK/CK FDs:
    - {Name, Pet ID} -> {Name, Pet ID} (PK -> all attributes)

Normalization

Staff

- Recall the function dependencies:
- PK/CK FDs:
    - Staff ID -> Name, Email, Address, Postal Code, City (CK-> all attributes)
    - Email -> Staff ID (CK -> all attributes)
- Non-PK/CK FDs:
    - Address -> Postal Code, City
- The only relation not in BCNF is the FD Address->Postal Code, City, as Address is not a superkey
    - After decomposing on Address->Postal Code, City, we get 2 tables:
        - Staff1(Name, Email, Address, Staff ID, Adopter ID) and Staff2(Address, Postal Code, City)
    - As there are no remaining relations violating BCNF, these are the final

relations

(Original)

```
Staff (
        Name : VARCHAR(50),
        Email : VARCHAR(255),
        Address : VARCHAR(255),
        Staff ID : INTEGER,
        Postal Code : VARCHAR(6),
        City : VARCHAR(20))
PRIMARY KEY(Staff ID),     <- PK
UNIQUE(Email)       <- CK
```

(Post Decomposition)

(Staff1)

```
Staff1 (
        Name : VARCHAR(50),
        Email : VARCHAR(255),
        Address : VARCHAR(255),
        Staff ID : INTEGER)
PRIMARY KEY(Staff ID),     <- PK
```

UNIQUE(Email)   <- CK

(Staff2)

Staff2 (

Address : VARCHAR(255),

Postal Code : VARCHAR(6),

City : VARCHAR(20))

PRIMARY KEY (Address),   <- PK

Adopter

- Recall the functional dependencies
- PK/CK FDs:
  - Adopter ID -> Name, Email, Address, Postal Code, City(PK -> all attributes)
  - Email -> Adopter ID (CK -> all attributes)
- Non-PK/CK FDs:
  - Address -> Postal Code, City
- The only relations not in BCNF are Total Duration, Start -> End and Start, End -> Shift Type. We decompose as follows
  - After decomposing on Address->Postal Code, City, we get 2 tables:
    - Adopter1(Name, Email, Address, AdopterID)
    - Adopter2(Address, City, Postal Code)
  - There are no remaining relations that violate BCNF, these are final relations

(Original)

Adopter(

Name : VARCHAR(50),

Email : VARCHAR(255),

Address : VARCHAR(255),

Adopter ID : INTEGER,

Postal Code : VARCHAR(6),

City : VARCHAR(20))

PRIMARY KEY(Adopter ID)   <- PK

UNIQUE(Email)   <- CK

(Post Decomposition)

(Adopter1)

Adopter1(

Name : VARCHAR(50),

Email : VARCHAR(255),

Address : VARCHAR(255),

Adopter ID : INTEGER)

PRIMARY KEY(Adopter ID),   <- PK

UNIQUE(Email)   <- CK

(Adopter2)

Adopter2 (

Address : VARCHAR(255),

Postal Code : VARCHAR(6),

City : VARCHAR(20))

PRIMARY KEY (Address),   <- PK

<u>Works_Shift</u>

- Recall the functional dependencies
- PK/CK FDs:
    - Date, Start, End -> Shift Type, Total Duration, Staff ID (PK -> Staff working the shift)
    - Staff ID -> Date, Start, End Shift Type, Total Duration (CK -> Staff working the shift)
- Non-PK/CK FDs:
    - Total Duration, Start -> End (for example, full-time or part time)
    - Start, End -> Shift Type
- The relations not in BCNF are:
    - Total Duration, Start -> End
    - Start, End -> Shift Type
- Decomposing on Total Duration, Start -> End gives us 2 tables:
        - Works_Shift1(Total Duration, Start, Shift Type, Date, <u>StaffID</u>)
        - Works_Shift2(<u>Start</u>, End, <u>Total Duration</u>)
    - Start, End -> Shift Type no longer violates BCNF, so these relations are final

(Original)

Works_Shift(

<u>Date</u> : DATE,

Total Duration : INTEGER,

<u>Start</u> : TIME,

Staff ID: VARCHAR(10),

Shift Type: VARCHAR(20))

PRIMARY KEY(Date, Start, End)   <- PK

FOREIGN KEY (Staff ID) REFERENCES Staff   <- FK

FOREIGN KEY (Shift Type) REFERENCES Shift   <- FK

(Post Decomposition)

(Works_Shift1)

Works_Shift1(

<u>Date</u> : DATE,

Total Duration : INTEGER,

<u>Start</u> : TIME,

<u>End</u> : TIME,

Staff ID: VARCHAR(10),

Shift Type: VARCHAR(20))

PRIMARY KEY(Staff ID)   <- PK

FOREIGN KEY (Shift Type) REFERENCES Shift   <- FK

(Works_Shift2)

Works_Shift2(

<u>Total Duration </u>: INTEGER,

<u>Start </u>: TIME,

End: TIME

PRIMARY KEY(Total Duration, Start)   <- PK

<u>Pet</u>

- Recall the functional dependencies
- PK/CK FDs:
    - PetID -> City, Status, Age, Name (PK -> all attributes)
- Non-PK/CK FDs:
    - City -> Status (Some pets may only be available for adoption in certain cities)

- The relations not in BCNF are:
  - City -> Status
- Decomposing on City -> Status gives us 2 tables:
  - Pet1(PetID, City, Age, Name)
  - Pet2(City, Status)

(Original)

Pet(

City : VARCHAR(20),

ID : INTEGER,

Status : BOOLEAN,

Age : INTEGER,

Name : VARCHAR(50))

PRIMARY KEY(ID) <- PK

(Post Decomposition)

(Pet1)

Pet1(

City : VARCHAR(20),

PetID : INTEGER,

Age : INTEGER,

Name : VARCHAR(50))

PRIMARY KEY(ID) <- PK

(Pet2)

Pet2(

City : VARCHAR(20),

Status : BOOLEAN,

PRIMARY KEY(City) <- PK

Vaccination

*More descriptive due to larger decompositions

These Relations are not in BNCF as the LHS are not super keys:

Vaccine Name -> Effectiveness Period (Vaccines have a known duration of effectiveness)

{Date, Effectiveness Period} -> Next Due

{Date, Next Due} -> Effectiveness Period

Decompose Vaccine Name -> Effectiveness Period

Vaccine1 (Vaccine Name, Vet Name, Next Due, Date, Vaccine ID)

Vaccine2 (Vaccine Name, Effectiveness Period)

Decompose {Date, Effectiveness Period} -> Next Due with S1

Vaccine3 (Vaccine ID, Vaccine Name, Vet Name, Date)

Vaccine4 (Date, Effectiveness Period, Next Due)

Decompose {Date, Next Due} -> Effectiveness Perioid WITH s4

Vaccine5 (Date, Next Due, Effectiveness Period)

The tables we need are

Vaccine1(Vaccine Name, Vet Name, Next Due, Date, Vaccine ID)

Vaccine2(<u>Vaccine Name</u>, Effectiveness Period)

Vaccine4(<u>Date, Effectiveness Perioid</u>, Next Due)

Relation Vaccine5 is redundant.

(Original)

Vaccination(Vaccine Name: VARCHAR(50),

Next Due: DATE,

Vet Name: VARCHAR(50),

Date: DATE,

<u>Vaccine ID</u>: INTEGER,

Effectiveness Period: INTEGER)

PRIMARY KEY(Vaccine ID) <- PK

(Post Decomposition)

(Vaccine1)

Vaccine1 (Vaccine ID : INTEGER,

Vaccine Name : VARCHAR(50),

Vet Name : VARCHAR(50),

Next Due : DATE, Date : DATE)

PRIMARY KEY(Vaccine ID)

(Vaccine2)

Vaccine2 (Vaccine Name : VARCHAR(50),

Effectiveness Period : INTEGER)

PRIMARY KEY(Vaccine Name)

(Vaccine3)

Vaccine3 (Date : DATE,

Effectiveness Period : INTEGER,

Next Due : DATE)

PRIMARY KEY(Date, Effectiveness Period)

Diet

*More descriptive due to larger decompositions

The only CK is Diet ID

All of these relations are BNCF Violations as the LHS are not superkeys

{Portion Size, Calories} -> Meal Frequency (Larger portion sizes mean less meals)

Allergies -> Restricted Foods (Certain allergies require avoiding specific foods)

{Avilable Foods, Restricted Foods} -> Chosen Foods

Chosen Foods -> Calories

Decompose {Portion Size, Calories} -> Meal Frequency

Diet1(Diet ID, Portion Size, Allergies, Restricted Foods, Calories, Chosen Foods, Available Foods)

Diet2(Portion Size, Calories, Meal Frequency) (PK: {Portion Size, Calories})

Decompose Allergies -> Restricted Foods with S1

Diet3(Diet ID, Portion Size, Allergies, Calories, Chosen Foods, Available Foods)

Diet4(Allergies, Restricted Foods) (PK: Allergies)

Decompose {Avilable Foods, Restricted Foods} -> Chosen Foods with S3

Diet5(Diet ID, Portion Size, Allergies, Calories, Available Foods)

Diet6(Available Foods, Restricted Foods, Chosen Foods) (PK: {Available Foods, Restricted Foods})

Decompose Chosen Foods -> Calories with S6

Diet7(Available Foods, Restricted Foods, Chosen Foods)

Diet8(Chosen Foods, Calories) (PK: Chosen Foods)

The Tables are
Diet1(<u>Diet ID</u>, Portion Size Allergies, Calories, Chosen Foods, Available Foods)
Diet(<u>Portion Size, Calories</u>, Meal Frequency)
Diet4(<u>Allergies</u>, Restricted Foods)
Diet7(<u>Available Foods, Restricted Foods</u>, Chosen Foods)

(Original)

Diet(<u>ID</u>: INTEGER,

Portion Size: DECIMAL(10,2),

Allergies: TEXT[])

PRIMARY KEY (ID) <- PK

(Post Decomposition)

(Diet1)

Diet1 (Diet ID : INTEGER,

Portion Size : DECIMAL(10,2),

Allergies : TEXT[],

Calories : TEXT[],

Chosen Foods: TEXT[],

Available Foods : TEXT[])

PRIMARY KEY(Diet ID) <- PK

(Diet2)

Diet2 (Portion Size : DECIMAL(10,2),

Calories : INTEGER,

Meal Frequency : INTEGER

PRIMARY KEY(Portion Size, Calories) <- PK

(Diet3)

Diet3 (Allergies : TEXT[],

Restricted Foods : TEXT[])

PRIMARY KEY(Allergies) <- PK

(Diet 4)

Diet4 (Available Foods : TEXT[],

Restricted Foods : TEXT[],

Chosen Foods : TEXT[])

PRIMARY KEY(Available Foods, Restricted Foods) <- PK

(Diet 5)

Diet5 (Chosen Foods : TEXT[],

Calories : INTEGER)

PRIMARY KEY(Chosen Foods) <- PK

SQL DDL statements

Staff

```
CREATE TABLE Staff1 (

   Staff_ID INTEGER PRIMARY KEY,

   Name VARCHAR(50) NOT NULL,

   Email VARCHAR(255) NOT NULL UNIQUE,

   Address VARCHAR(255) NOT NULL

);


CREATE TABLE Staff2 (

   Address VARCHAR(255) PRIMARY KEY,

   Postal_Code VARCHAR(6) NOT NULL,
    City VARCHAR(20) NOT NULL
);
```

Adopter

```
CREATE TABLE Adopter1 (
    Adopter_ID INTEGER PRIMARY KEY,
    Name VARCHAR(50) NOT NULL,
    Email VARCHAR(255) NOT NULL UNIQUE,
    Address VARCHAR(255) NOT NULL
);

CREATE TABLE Adopter2 (
    Address VARCHAR(255) PRIMARY KEY,
    Postal_Code VARCHAR(6) NOT NULL,
    City VARCHAR(20) NOT NULL
);
```

Works_Shift

```
CREATE TABLE Works_Shift1 (
    Date DATE NOT NULL,
    Total_Duration INTEGER NOT NULL,
    Start TIME NOT NULL,
    End TIME NOT NULL,
    Staff_ID VARCHAR(10) NOT NULL,
    Shift_Type VARCHAR(20) NOT NULL,
    PRIMARY KEY (Date, Start, Staff_ID),
    FOREIGN KEY (Staff_ID) REFERENCES Staff(Staff_ID),
    FOREIGN KEY (Shift_Type) REFERENCES Shift(Shift_Type)
);


CREATE TABLE Works_Shift2 (
    Total_Duration INTEGER NOT NULL,
    Start TIME NOT NULL,
    End TIME NOT NULL,
    PRIMARY KEY (Total_Duration, Start)
);
```

<u>Pet</u>

```
CREATE TABLE Pet1 (
    PetID INTEGER PRIMARY KEY,
    City VARCHAR(20) NOT NULL,
    Age INTEGER NOT NULL,
    Name VARCHAR(50) NOT NULL,
    FOREIGN KEY (City) REFERENCES Pet2(City)
);

CREATE TABLE Pet2 (
    City VARCHAR(20) PRIMARY KEY,
    Status BOOLEAN NOT NULL
);
```

Vaccine

```
CREATE TABLE Vaccine1 (
    Vaccine_ID INTEGER PRIMARY KEY,
    Vaccine_Name VARCHAR(50) NOT NULL,
    Vet_Name VARCHAR(50) NOT NULL,
    Next_Due DATE NOT NULL,
    Date DATE NOT NULL,
    FOREIGN KEY (Vaccine_Name) REFERENCES Vaccine2(Vaccine_Name)
);

CREATE TABLE Vaccine2 (
    Vaccine_Name VARCHAR(50) PRIMARY KEY,
    Effectiveness_Period INTEGER NOT NULL CHECK (Effectiveness_Period > 0)
);
CREATE TABLE Vaccine3 (
    Date DATE NOT NULL,
    Effectiveness_Period INTEGER NOT NULL CHECK (Effectiveness_Period > 0),
    Next_Due DATE NOT NULL,
    PRIMARY KEY (Date, Effectiveness_Period)
);
```

Diet

```
CREATE TABLE Diet1 (
    Diet_ID INTEGER PRIMARY KEY,
    Portion_Size DECIMAL(10,2) NOT NULL,
    Allergies TEXT[] NOT NULL,
    Calories TEXT[] NOT NULL,
    Chosen_Foods TEXT[] NOT NULL,
    Available_Foods TEXT[] NOT NULL
);
CREATE TABLE Diet2 (
    Portion_Size DECIMAL(10,2) NOT NULL,
    Calories INTEGER NOT NULL,

    Meal_Frequency INTEGER NOT NULL,

    PRIMARY KEY (Portion_Size, Calories)

);

CREATE TABLE Diet3 (

    Allergies TEXT[] PRIMARY KEY,

    Restricted_Foods TEXT[] NOT NULL

);


CREATE TABLE Diet4 (

    Available_Foods TEXT[] NOT NULL,

    Restricted_Foods TEXT[] NOT NULL,

    Chosen_Foods TEXT[] NOT NULL,

    PRIMARY KEY (Available_Foods, Restricted_Foods)

);

CREATE TABLE Diet5 (

    Chosen_Foods TEXT[] PRIMARY KEY,

    Calories INTEGER NOT NULL

);
```

INSERT statements

Staff1 & Staff2:

```
INSERT INTO Staff1 (Staff_ID, Name, Email, Address)

VALUES

   (1, 'Morgan Freeman', 'morganf@gmail.com', '100 Hollywood Blvd'),

   (2, 'Zendaya', 'zendaya@yahoo.com', '750 Everglades Terrace'),

   (3, 'Robert Downey Jr.', 'rdj@yahoo.com', '177A Blob Street'),

   (4, 'Emma Watson', 'emmaw@skype.com', '13 Grim Place'),

   (5, 'Dwayne Johnson', 'therock@gmail.com', '123 Mcknight Road');

INSERT INTO Staff2 (Address, Postal_Code, City)

VALUES

   ('100 Hollywood Blvd', '90028', 'Los Angeles'),

   ('750 Everglades Terrace', 'V5K0A1', 'Springfield'),

   ('177A Blob Street', '10012', 'New York'),

   ('13 Grim Place', 'EC1A 1BB', 'London'),

   ('123 Mcknight Road', '32830', 'Orlando');
```

Adopter1 & Adopter2:

```
INSERT INTO Adopter1 (Adopter_ID, Name, Email, Address)

VALUES

    (1, 'Tom Hanks', 'tomh@skype.com', '742 Everglades Terrace'),

    (2, 'Scarlett Johansson', 'scarlett@yahoo.com', '221B Bourne Street'),

    (3, 'Ryan Reynolds', 'ryanr@microsoft.com', '1600 Penn Ave'),

    (4, 'Taylor Swift', 'taylors@gmail.com', '12 Grim Place'),

    (5, 'Keanu Reeves', 'keanur@gmail.com', '10 Down Street');


INSERT INTO Adopter2 (Address, Postal_Code, City)

VALUES

    ('742 Everglades Terrace', 'V5K0A1', 'Springfield'),

    ('221B Bourne Street', 'NW1 6XE', 'London'),

    ('1600 Pennsylvania Ave', '20500', 'Washington DC'),

    ('12 Grim Place', 'EC1A 1BB', 'London'),

    ('10 Down Street', 'SW1A 2AA', 'London');
```

Works_Shift1 & Works_Shift2

```
INSERT INTO Works_Shift1 (Date, Total_Duration, Start, End, Staff_ID, Shift_Type)

VALUES

    ('2025-03-01', 8, '09:00:00', '17:00:00', 1, 'Morning Shift'),

    ('2025-03-01', 6, '12:00:00', '18:00:00', 2, 'Afternoon Shift'),
    ('2025-03-02', 4, '18:00:00', '22:00:00', 3, 'Evening Shift'),
    ('2025-03-03', 5, '10:00:00', '15:00:00', 4, 'Afternoon Shift'),
    ('2025-03-04', 10, '08:00:00', '18:00:00', 5, 'Full-time Shift');

INSERT INTO Works_Shift2 (Total_Duration, Start, End)
VALUES
    (8, '09:00:00', '17:00:00'),
    (6, '12:00:00', '18:00:00'),
    (4, '07:00:00', '11:00:00'),
    (5, '15:00:00', '20:00:00'),
    (7, '22:00:00', '05:00:00');
```

Pet1 & Pet2

```
INSERT INTO Pet1 (PetID, City, Age, Name)
VALUES
    (1, 'Los Angeles', 2, 'Bark Wahlberg'),
    (2, 'New York', 4, 'Meowly Cyrus'),
    (3, 'Chicago', 1, 'Leonardo DiCatrio'),
    (4, 'Toronto', 5, 'Howl Jackman'),
    (5, 'London', 3, 'Woofie Goldberg');
```

```
INSERT INTO Pet2 (City, Status)

VALUES

    ('Los Angeles', TRUE),

    ('New York', FALSE),

    ('Chicago', TRUE),

    ('Toronto', TRUE),

    ('London', FALSE);
```

Vaccination

```
INSERT INTO Vaccine1 (Vaccine_ID, Vaccine_Name, Vet_Name, Next_Due, Date)

VALUES

(101, 'Rabies Vaccine', 'Dr. Smith', '2025-06-15', '2024-06-15'),

(102, ‘Polio Vaccine', 'Dr. Johnson', '2025-07-20', '2024-07-20'),

(103, ‘Malaria Vaccine', 'Dr. Brown', '2025-08-10', '2024-08-10');

(104, Smallpox Vaccine', 'Dr. Wang, '2025-09-11', '2024-09-11');

(105, Tetnis Vaccine', 'Dr. Joe, '2025-10-12', '2024-10-12');


INSERT INTO Vaccine2 (Vaccine_Name, Effectiveness_Period)

VALUES

('Rabies Vaccine', 12),
(‘Polio Vaccine', 12),

(‘Malaria Vaccine', 12);

(Smallpox Vaccine', 12);

(Tetnis Vaccine', 12);


INSERT INTO Vaccine3 (Date, Effectiveness_Period, Next_Due)

VALUES

('2024-06-15', 12, '2025-06-15'),

 ('2024-07-20', 12, '2025-07-20'),

('2024-08-10', 12, '2025-08-10')

('2024-09-11', 12, '2025-09-11')

('2024-10-12', 12, '2025-10-12')
```

Diet

```
INSERT INTO Diet1 (Diet_ID, Portion_Size, Allergies, Calories, Chosen_Foods,
Available_Foods)

VALUES

(1, 250, ARRAY['Gluten'], ARRAY['500'], ARRAY['Chicken', 'Rice'], ARRAY['Chicken',
'Rice', 'Bread']),

(2, 200, ARRAY['Dairy'], ARRAY['450'], ARRAY['Salmon', 'Broccoli'], ARRAY['Salmon',
'Broccoli', 'Milk']),

(3, 300, ARRAY['Peanuts'], ARRAY['600'], ARRAY['Eggs', 'Spinach'], ARRAY['Eggs',
'Spinach', 'Peanut Butter']),
(4, 275, ARRAY['Soy'], ARRAY['550'], ARRAY['Beef', 'Carrots'], ARRAY['Beef', 'Carrots',
'Soy Sauce']),

(5, 225, ARRAY['Seafood'], ARRAY['480'], ARRAY['Lentils', 'Sweet Potato'],
ARRAY['Lentils', 'Sweet Potato', 'Shrimp']);


INSERT INTO Diet2 (Portion_Size, Calories, Meal_Frequency)

VALUES

(250, 500, 3),

(200, 450, 4),

(300, 600, 2),

(275, 550, 3),

(225, 480, 3);


INSERT INTO Diet3 (Allergies, Restricted_Foods)

VALUES

(ARRAY['Gluten'], ARRAY['Bread', 'Pasta']),

(ARRAY['Dairy'], ARRAY['Milk', 'Cheese']),

(ARRAY['Peanuts'], ARRAY['Peanut Butter', 'Peanuts']),

(ARRAY['Soy'], ARRAY['Soy Sauce', 'Tofu']),

(ARRAY['Seafood'], ARRAY['Shrimp', 'Fish']);


INSERT INTO Diet4 (Available_Foods, Restricted_Foods, Chosen_Foods)
```

```
VALUES

(ARRAY['Chicken', 'Rice', 'Bread'], ARRAY['Bread'], ARRAY['Chicken', 'Rice']),

(ARRAY['Salmon', 'Broccoli', 'Milk'], ARRAY['Milk'], ARRAY['Salmon', 'Broccoli']),

(ARRAY['Eggs', 'Spinach', 'Peanut Butter'], ARRAY['Peanut Butter'], ARRAY['Eggs',
'Spinach']),

(ARRAY['Beef', 'Carrots', 'Soy Sauce'], ARRAY['Soy Sauce'], ARRAY['Beef', 'Carrots']),

(ARRAY['Lentils', 'Sweet Potato', 'Shrimp'], ARRAY['Shrimp'], ARRAY['Lentils', 'Sweet
Potato']);


INSERT INTO Diet5 (Chosen_Foods, Calories)

VALUES

(ARRAY['Chicken', 'Rice'], 500),

(ARRAY['Salmon', 'Broccoli'], 450),

(ARRAY['Eggs', 'Spinach'], 600),

(ARRAY['Beef', 'Carrots'], 550),

(ARRAY['Lentils', 'Sweet Potato'], 480);
```

AI Acknowledgement:

We certify that the work in this document is our own, and we did not use AI tools in assistance for generation of this document.