Project 1

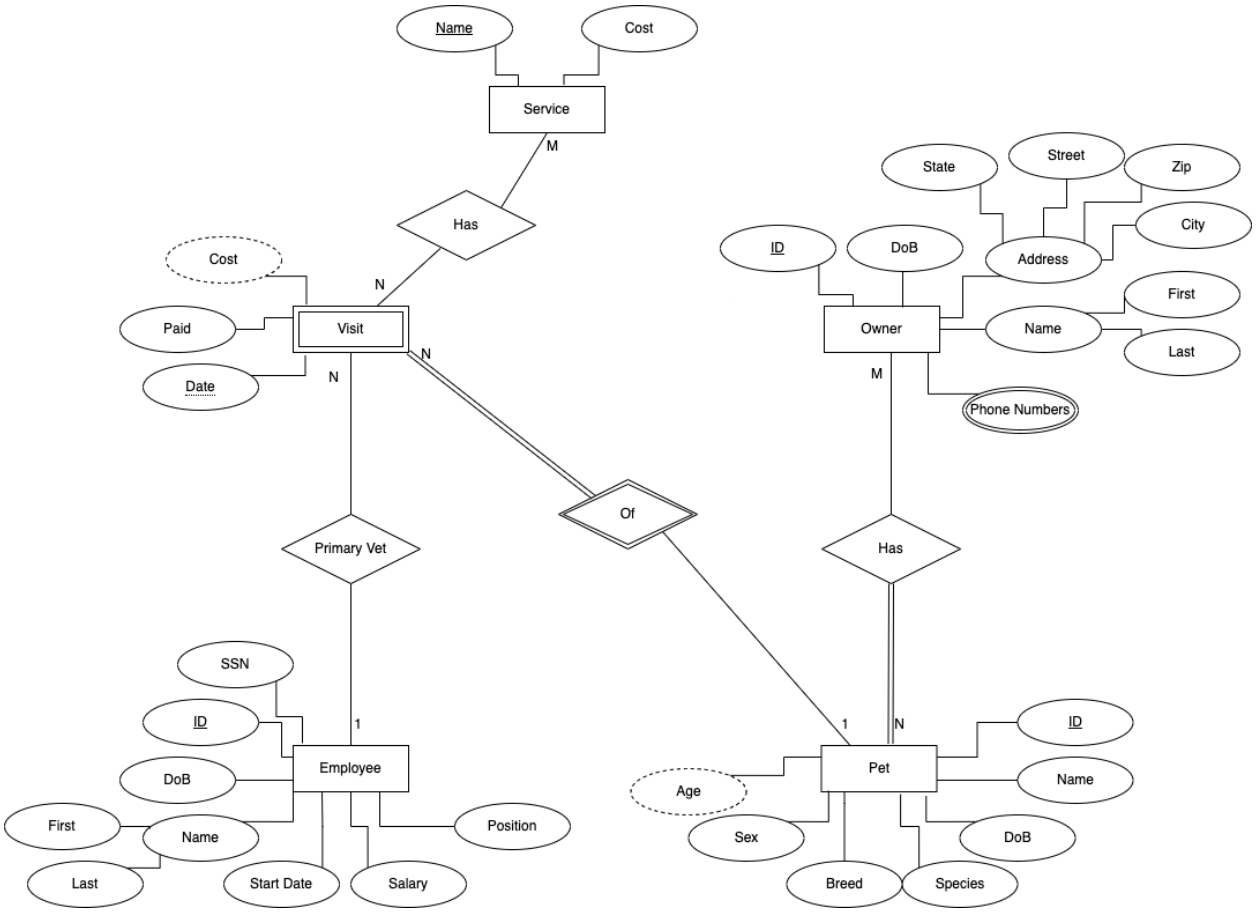
1. Overview

The goal of this database is to serve a single vets office. In this, we want to be able to track employees that work at this vets office, pet owners, pets, and the visits of these pets. In this, we want to be able to track things such as pet visit history, whether or not those visits have been paid for, and other essential services that a vets office might require.

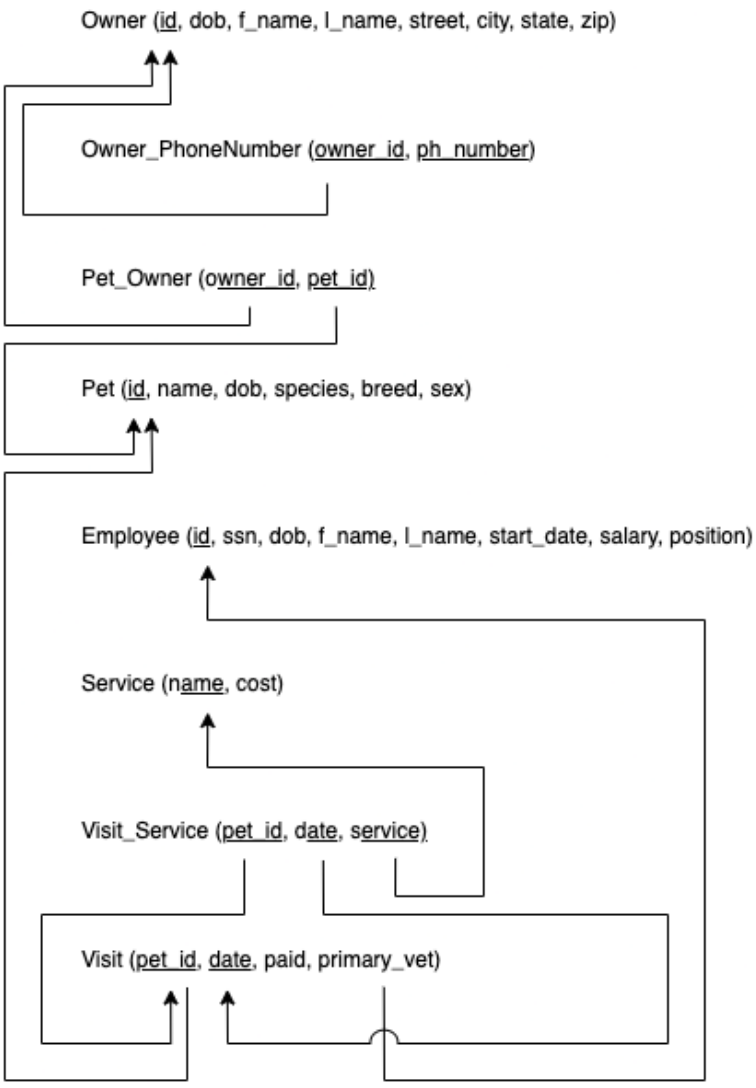
For the employees, we want to be able to identify them with a unique ID that is auto generated when they are first hired, and store their SSN, salary, name, date of birth, position, and start date. For the owners, they are also identified by a unique ID that is auto generated on their first visit, and we want to store their name, date of birth, address, any phone numbers they might have, and their pets. The pets are also identified by a unique ID that is auto generated on their first visit, and we want to store their name, date of birth, their species, breed, sex, and age. We also want these pets to be able to have multiple owners. Each visit that a pet has (limitted to one visit per pet per day), has a the pet that was visiting, the visit date, the primary vet, and whether or not payment has been completed for this visit. Note, We will not be storing any payment information, only that a payment has, or has not, been completed. The vets office also has a preset list of services with unique names that all have an associated cost.

When an owner first shows up to the office, they provide their info (see above), and can either enter in a new pet, or they can associate themselves with an existing pet in the database. Similarly, when an employee is first hired, their info is entered in, and mostly cannot be changed. Though, they can receive a pay raise/decrease, as long as the new salary is greater than 0. During a visit, a vet will take note of which services were given and later an employee will save which services were given (entered in individually, where duplicate services are not allowed). Then, for clerical purposes, an employee can find a list of visits that have not been paid for, the cost of each unpaid visit (separately), get all owners with their contact information and pet information, get a list of all owners that do not have contact information (at least one phone number), get a list of pets that have never had a visit, get a list of pets with multiple owners. For more statistical use, they can also get an ordered list of how many visits happen per species. And if they are feeling especially generous, an employee can give anyone who makes less than $70,000 a 5% raise.

2. Entity-Relationship Diagram



3. Relational Schema



4. Tables

Table 1: Owner
Purpose: Tracks the information of owners (aka clients) that bring their pets into the office.
Attributes: id, dob, f_name, l_name, street, city, state, zip
Primary Key: id (INT UNSIGNED AUTO_INCREMENT)

When owners are added, id should not be specified. They are auto generated when a new owner is added.

```
CREATE TABLE Owner (
        id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
        dob Date,
        f_name varchar(30),
        l_name varchar(30),
        street varchar(40),
        city varchar(20),
        state char(2),
        zip char(5)
);
```

Table 1: Owner

id	dob	f_name	l_name	street	city	state	zip
1	1946-02-14	Selena	Gomez	1234 Example St.	Los Angeles	CA	94928
2	1897-12-24	Hailey	Beiber	4321 Other St.	Santa Rosa	CA	94928
3	2001-01-01	Mr.	Worldwide	1 Main St.	Miami	FL	33101
4	1990-02-02	Jonathon	Thompson	32 West St	San Francisco	CA	12345
5	1984-05-15	Charles E	Cheese	3165 Heavens Way	Boca Grande	FL	33921
6	1973-03-24	Tobey	Maguire	2953 Jett Lane	Pomona	CA	91766
7	2002-09-12	Olzoneth	Rilore	2040 Romrog Way	Grand Island	NE	68801
8	1999-10-31	Jack	Skellington	1 Halloween Way	Halloween Town	ND	43215
9	1986-09-30	Barbara	Matel	1959 Malibu Way	Malibu	CA	90263
10	1997-04-01	Winnie	The Pooh	916 Gateway Avenue	Bakersfield	CA	93301
11	1989-12-01	Phillip	Fry	3230 Alexander Avenue	San Jose	CA	95131
12	1974-11-11	Scooby	Doo	4898 Thunder Road	Millbrae	CA	94030
13	1973-08-08	Darkwing	Duck	2111 Carson Street	La Mesa	CA	91941
14	1990-04-10	Lizzie	McGuire	3003 Rosemont Avenue	Los Angeles	CA	90017
15	1984-06-17	Allister	Shinigami	1 Main St.	Miami	FL	33101

Table 2: Pet
Purpose: To store the information of a single pet
Attributes: id, name, dob, species, breed, sex
Primary Key: id (INT UNSIGNED AUTO_INCREMENT)

When pets are added, id should not be specified. They are auto generated when a new pet is added.

```
CREATE TABLE Pet (
        id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
        name varchar(35),
        dob DATE,
        species  ENUM ('dog', 'cat', 'bird', 'snake', 'hamster', 'guinea pig', 'fish', 'turtle') NOT
NULL,
        breed varchar(20),
        sex ENUM ('male', 'female', 'other')
);
```

Table 2: Pet

id	name	dob	species	breed	sex
1	Barksley	2012-01-01	dog	mastiff	male
2	Captain Waffles	2017-09-03	cat	siamese	male
3	Car Seat French Fry	2011-11-21	cat	ragdoll	male
4	Dad	2007-07-17	hamster	BREED	male
5	Scissor Bill	2008-10-25	dog	bulldog	other
6	Special Agent Dale Cooper FBI	2008-07-20	cat	tabby	other
7	Mr. Ugly	2019-12-06	dog	pitbull	female
8	Lotion	2016-01-09	fish	goldfish	male
9	Lorde	2013-03-27	snake	python	male
10	HOME DEPOT	2020-07-10	guinea pig	american	male
11	Bratwurst	2002-04-26	hamster	syrian	male
12	McRib	2018-11-11	dog	german shepard	other
13	Milk	2018-09-07	cat	tabby	other
14	Old Cat	2008-05-29	cat	tabby	male
15	Sour Cream	2017-07-07	dog	husky	female
16	Cher	2007-12-14	cat	tabby	male
17	Mr Tumnus	2003-01-21	cat	short hair	female
18	Potato	2008-02-07	fish	koi	other
19	Pound Cake	2009-04-17	bird	parrot	other
20	King Gary	2008-11-29	hamster	chinese	other
21	Dorito	2004-09-12	fish	koi	other
22	Aunt Bethany	2007-10-20	snake	copperhead	male
23	Soup	2014-06-15	dog	mutt	other
24	Leondardo DogVinci	2015-11-12	dog	border collie	female
25	Feet	2002-12-08	dog	pomeranian	other
26	Dracula	2019-07-27	cat	shorthair	other
27	Stinky Chunky	2018-12-01	cat	tabby	female
28	Tom	2020-01-01	turtle	spotted	male

Table 3: Service
Purpose: To store the services that this vets office offers.
Attributes: name, cost
Primary Key: name (varchar(30))

```
CREATE TABLE Service (
        name varchar(30) PRIMARY KEY,
        cost numeric (8, 2) CHECK (cost >= 0)
);
```

Table 3: Service

name	cost
Allergy testing	225.00
Bloodwork	115.00
Dental cleaning	250.00
Emergency surgery	3500.00
Fecal exam	35.00
Geriatric screening	105.00
Heartworm test	55.00
Long hospitalization	2750.00
Nail Trim	15.00
Oxygen therapy	2500.00
Physical exam	50.00
Routine checkup	100.00
Short hospitalization	1200.00
Single Vaccine	50.00
Spay/neuter	180.00
Ultrasound	400.00
Wound treatment	1150.00
X-ray	165.00

Table 4: Employee
Purpose: Stores the information of a single employee at this vets office
Attributes: id, ssn, dob, f_name, l_name, start_date, salary, position
Primary Key: id (INT UNSIGNED AUTO_INCREMENT)

When employees are added, id should not be specified. They are auto generated when a new employee is added.

```
CREATE TABLE Employee (
        id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
        ssn varchar(9) UNIQUE NOT NULL,
        dob DATE,
        f_name varchar(30),
        l_name varchar(30),
        start_date DATE,
        salary numeric (12, 2) CHECK (salary > 0),
        position varchar(20)
);
```

Table 4: Employee

id	ssn	dob	f_name	l_name	start_date	salary	position
1	663403032	1984-06-13	Ashley	Katchadorian	2016-06-28	113863.00	Vet Assistant
2	114806854	1992-12-07	Eren	Yaegar	2007-05-17	128416.00	Veterinarian
3	426744790	1980-07-27	Britney	Mathews	2013-10-10	40133.00	Intern
4	166181536	1983-12-06	Mackenzie	Zales	2002-07-04	176559.00	Veterinarian
5	558712500	1987-03-20	Trisha	Cappalletti	2016-04-23	98758.00	Vet Assistant
6	933687702	1965-05-01	Clifford	Walsh	1993-04-02	71903.00	Vet Tech
7	775191066	1973-08-26	Patsy	Hawkins	2013-03-23	64633.00	Janitor
8	851865416	1975-09-25	Eulau	Choi	1988-10-09	65211.00	Receptionist
9	261385255	1963-05-16	Ernest	Huang	1997-07-14	75268.00	Vet Assistant
10	796972898	1971-11-14	Elisa	Capps	1995-02-01	63000.00	Receptionist

Table 5: Visit
Purpose: Tracks the visit of a pet at this vets office on a single day
Attributes: pet_id, date (of visit), paid, primary_vet
Primary Key: pet_id (INT UNSIGNED), date (DATE)
Foreign Key(s): pet_id references Pet.id – primary_vet references Employee.id

```
CREATE TABLE Visit (
        pet_id INT UNSIGNED,
        date DATE,
        paid BOOL DEFAULT false,
        primary_vet INT UNSIGNED,
        PRIMARY KEY (pet_id, date),
        FOREIGN KEY (pet_id) REFERENCES Pet(id)
                ON DELETE NO ACTION
                ON UPDATE CASCADE,
        FOREIGN KEY (primary_vet) REFERENCES Employee(id)
                ON DELETE NO ACTION
                ON UPDATE CASCADE
);
```

Table 5: Visit

pet_id	date	paid	primary_vet
1	2000-12-16	0	4
3	2018-01-15	0	2
4	2018-08-27	0	2
5	1994-05-06	1	4
6	1989-02-28	1	2
6	1990-05-02	0	4
6	2008-05-12	0	4
8	1998-08-29	0	2
9	2012-01-19	0	4
9	2012-01-20	0	NULL
10	2002-11-30	1	2
12	2014-01-08	0	2
13	1986-08-22	1	4
15	2015-08-10	0	2
16	1988-04-22	1	2
17	2016-01-19	1	2
18	2005-03-07	0	2
19	1990-05-10	1	4
20	1991-09-03	0	4
20	2009-09-23	1	2
20	2010-08-24	0	2
22	1997-04-27	1	4
22	2019-10-01	1	2
23	1997-08-04	0	4
26	2000-12-15	1	4
27	2002-12-27	1	2
28	2020-10-10	0	NULL

Table 6: Owner_PhoneNumber
Purpose: Stores the phone number(s) for each owner in the database
Attributes: owner_id, ph_number
Primary Key: ph_number (INT UNSIGNED), ph_number varchar(13)
Foreign Key(s): owner_id references Owner.id

```
CREATE TABLE Owner_PhoneNumber (
        owner_id INT UNSIGNED,
        ph_number varchar(13),
        PRIMARY KEY (owner_id, ph_number),
        FOREIGN KEY (owner_id) REFERENCES Owner(id)
                ON DELETE CASCADE
                ON UPDATE CASCADE
);
```

Table 6: Owner_PhoneNumber

owner_id	ph_number
1	207-924-3726
1	209-988-1027
1	310-568-1308
1	310-897-1484
1	949-763-7785
2	209-853-6329
2	310-432-8940
2	918-385-8325
3	209-594-2906
3	209-663-8222
3	209-799-4298
3	918-282-6842
4	209-258-1543
4	209-839-9649
4	209-873-1643
5	209-654-7912
5	209-874-3600
6	310-547-6301
6	310-627-6664
6	417-623-8127
7	209-838-8587
7	310-555-9467
7	310-926-6780
7	585-644-6193
7	765-948-1670
8	209-476-3130
8	209-679-6425
8	209-825-8254
8	209-933-3435
9	209-948-2245
10	209-265-7087
10	209-659-5897
10	310-496-7239
10	310-765-1917
11	209-674-8765
11	209-695-7156
11	209-756-2675
11	310-884-7242
11	424-665-7388
12	310-338-8778
12	310-376-6925
12	310-379-6565
12	310-665-5830
13	843-725-8977
14	209-239-6584
14	209-563-4150
14	209-942-1040
14	310-489-2015
14	530-347-5007
14	971-537-4588

Table 7: Pet_Owner
Purpose: Stores the information of which owners own which pets
Attributes: owner_id, pet_id
Primary Key: owner_id (INT UNSIGNED), pet_id (INT UNSIGNED)
Foreign Key(s): owner_id references Owner.id — pet_id references Pet.id

```
CREATE TABLE Pet_Owner (
        owner_id INT UNSIGNED,
        pet_id INT UNSIGNED,
        PRIMARY KEY (owner_id, pet_id),
        FOREIGN KEY (owner_id) REFERENCES Owner(id)
                ON DELETE CASCADE
                ON UPDATE CASCADE,
        FOREIGN KEY (pet_id) REFERENCES Pet(id)
                ON DELETE CASCADE
                ON UPDATE CASCADE
);
```

Table 7: Pet_Owner

owner_id	pet_id
1	1
1	15
2	2
2	16
3	3
3	17
4	4
4	18
5	1
5	5
5	19
6	6
6	7
6	20
7	6
7	7
7	21
8	8
8	22
9	9
9	23
9	27
10	10
10	24
11	11
11	25
12	12
12	26
13	13
14	9
14	14
15	28

Table 8: Visit_Service
Purpose: Stores the list of services that were done for each visit
Attributes: pet_id, date, service
Primary Key: pet_id (INT UNSIGNED), date (DATE), service (VARCHAR(30))
Foreign Key(s): pet_id, date references Visit.pet_id, Visit.date – service references Service.name

```
CREATE TABLE Visit_Service (
        pet_id INT UNSIGNED,
        date DATE,
        service VARCHAR(30),
        PRIMARY KEY (pet_id, date, service),
        FOREIGN KEY (pet_id, date) REFERENCES Visit(pet_id, date)
                ON DELETE CASCADE
                ON UPDATE CASCADE,
        FOREIGN KEY (service) REFERENCES Service(name)
                ON DELETE NO ACTION
                ON UPDATE CASCADE
);
```

Table 8: Visit_Service

pet_id	date	service
1	2000-12-16	Dental cleaning
3	2018-01-15	Routine checkup
4	2018-08-27	Heartworm test
4	2018-08-27	X-ray
5	1994-05-06	Fecal exam
5	1994-05-06	Physical exam
5	1994-05-06	Routine checkup
6	1989-02-28	Ultrasound
6	1990-05-02	Routine checkup
6	2008-05-12	Long hospitalization
8	1998-08-29	Oxygen therapy
9	2012-01-19	Physical exam
10	2002-11-30	Nail Trim
10	2002-11-30	Routine checkup
12	2014-01-08	Dental cleaning
13	1986-08-22	Routine checkup
15	2015-08-10	Physical exam
15	2015-08-10	Spay/neuter
16	1988-04-22	Allergy testing
17	2016-01-19	Geriatric screening
18	2005-03-07	Spay/neuter
19	1990-05-10	Routine checkup
20	1991-09-03	Single Vaccine
20	2009-09-23	Wound treatment
20	2010-08-24	Emergency surgery
22	1997-04-27	Physical exam
22	2019-10-01	Spay/neuter
23	1997-08-04	Allergy testing
26	2000-12-15	Short hospitalization
27	2002-12-27	Routine checkup
28	2020-10-10	Physical exam

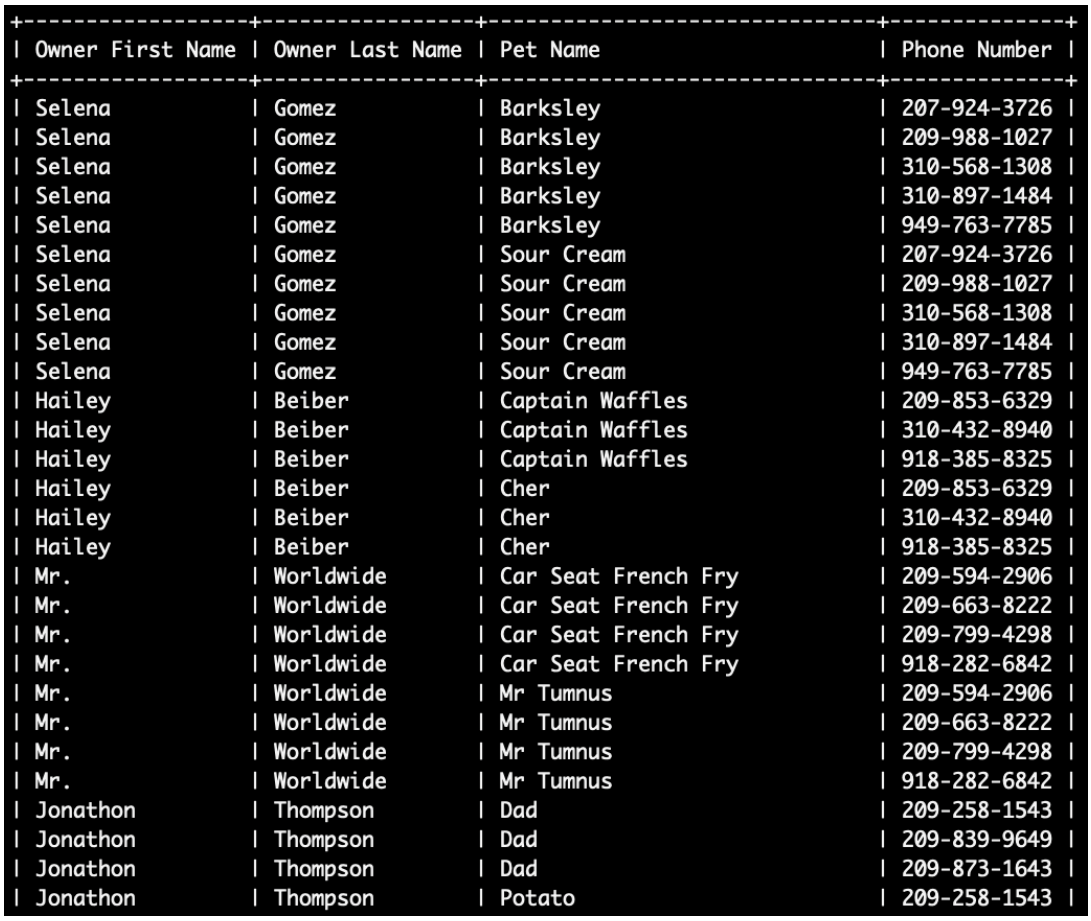
5. Queries

Query 1:
Lists all of the services that are offered by the vets office, but have never been used in a single visit.

```
SELECT name AS 'Unused Services'
FROM (
        (SELECT name FROM Service s)
        EXCEPT
        (SELECT service FROM Visit_Service vs)
) AS t;
```

Query 2:
Lists the contact information (phone number) of all owners with their pets.

```
SELECT
        o.f_name AS 'Owner First Name',
        o.l_name AS 'Owner Last Name',
        p.name AS 'Pet Name',
        opn.ph_number AS 'Phone Number'
FROM Pet p JOIN Pet_Owner po ON p.id = po.pet_id
        JOIN Owner o ON po.owner_id = o.id
        JOIN Owner_PhoneNumber opn ON opn.owner_id = o.id;
```

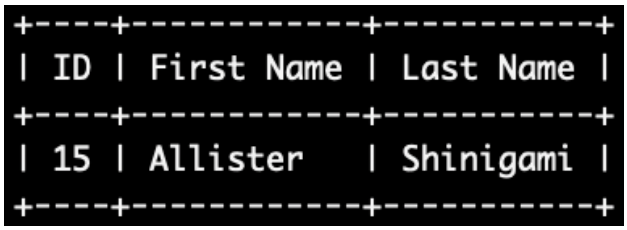


Owner First Name	Owner Last Name	Pet Name	Phone Number
Selena	Gomez	Barksley	207-924-3726
Selena	Gomez	Barksley	209-988-1027
Selena	Gomez	Barksley	310-568-1308
Selena	Gomez	Barksley	310-897-1484
Selena	Gomez	Barksley	949-763-7785
Selena	Gomez	Sour Cream	207-924-3726
Selena	Gomez	Sour Cream	209-988-1027
Selena	Gomez	Sour Cream	310-568-1308
Selena	Gomez	Sour Cream	310-897-1484
Selena	Gomez	Sour Cream	949-763-7785
Hailey	Beiber	Captain Waffles	209-853-6329
Hailey	Beiber	Captain Waffles	310-432-8940
Hailey	Beiber	Captain Waffles	918-385-8325
Hailey	Beiber	Cher	209-853-6329
Hailey	Beiber	Cher	310-432-8940
Hailey	Beiber	Cher	918-385-8325
Mr.	Worldwide	Car Seat French Fry	209-594-2906
Mr.	Worldwide	Car Seat French Fry	209-663-8222
Mr.	Worldwide	Car Seat French Fry	209-799-4298
Mr.	Worldwide	Car Seat French Fry	918-282-6842
Mr.	Worldwide	Mr Tumnus	209-594-2906
Mr.	Worldwide	Mr Tumnus	209-663-8222
Mr.	Worldwide	Mr Tumnus	209-799-4298
Mr.	Worldwide	Mr Tumnus	918-282-6842
Jonathon	Thompson	Dad	209-258-1543
Jonathon	Thompson	Dad	209-839-9649
Jonathon	Thompson	Dad	209-873-1643
Jonathon	Thompson	Potato	209-258-1543

Note: Parts of query output were cut off in the above screenshot as this query returned 110 rows which did not fit in the report.

Query 3:
Gets a list of all owners that do not have at least one phone number in the system.

```
SELECT
        ID,
        f_name AS 'First Name',
        l_name AS 'Last Name'
FROM Owner o LEFT JOIN Owner_PhoneNumber opn ON o.id = opn.owner_id
WHERE opn.ph_number IS NULL;
```



ID	First Name	Last Name
15	Allister	Shinigami

Query 4:
Gets the information of all pets that have multiple owners.

```
SELECT *
FROM Pet pout JOIN (
        SELECT
                p.id AS 'Pet ID',
                COUNT(*) AS 'Number of Owners'
        FROM Pet p JOIN Pet_Owner po ON p.id = po.pet_id
        GROUP BY p.id
        HAVING COUNT(*) > 1
) AS t ON pout.id = t.`Pet ID`;
```

id	name	dob	species	breed	sex	Pet ID	Number of Owners
1	Barksley	2012-01-01	dog	mastiff	male	1	2
6	Special Agent Dale Cooper FBI	2008-07-20	cat	tabby	other	6	2
7	Mr. Ugly	2019-12-06	dog	pitbull	female	7	2
9	Lorde	2013-03-27	snake	python	male	9	2

Query 5:
Gets an ordered list (descending order) of the total number of visits per species.

```
SELECT
        Species,
        COUNT(*) AS 'Number of Visits'
FROM Pet p
WHERE EXISTS (
        SELECT *
        FROM Visit v
        WHERE p.id = v.pet_id
)
GROUP BY p.species
ORDER BY COUNT(*) DESC;
```

Species	Number of Visits
cat	7
dog	5
hamster	2
fish	2
snake	2
guinea pig	1
bird	1
turtle	1

Query 7:
Gives any employee whose salary is less than $70,000 a 5% raise.

```
UPDATE Employee
        SET salary = salary * 1.05
        WHERE salary < 70000.00;
```

```
MariaDB [bdale_cs355sp23]> UPDATE Employee
    -> SET salary = salary * 1.05
    -> WHERE salary < 70000.00;
Query OK, 4 rows affected (0.001 sec)
Rows matched: 4  Changed: 4  Warnings: 0
```

Query 8:
Gets all pets that have not had a single visit

```
SELECT *
FROM Pet p
WHERE p.id NOT IN (
        SELECT DISTINCT v.pet_id
        FROM Visit v
);
```

id	name	dob	species	breed	sex
2	Captain Waffles	2017-09-03	cat	siamese	male
7	Mr. Ugly	2019-12-06	dog	pitbull	female
11	Bratwurst	2002-04-26	hamster	syrian	male
14	Old Cat	2008-05-29	cat	tabby	male
21	Dorito	2004-09-12	fish	koi	other
24	Leondardo DogVinci	2015-11-12	dog	border collie	female
25	Feet	2002-12-08	dog	pomeranian	other

6. Views, Function, and Procedures

View 1: vw_OwnersOutstandingPayments

View gets the id, pets, and name (first and last) of all owners who have outstanding payments. Can be used in conjunction with the function to get cost remaining to determine who owes money, and then how much they owe.

```
CREATE OR REPLACE VIEW vw_OwnersOutstandingPayments AS
(
	SELECT
		owner_id AS 'Owner ID',
		pet_id AS 'Pet ID',
		f_name AS 'Owner First Name',
		l_name AS 'Owner Last Name'
	FROM Pet_Owner po JOIN Owner o ON po.owner_id = o.id
	WHERE EXISTS (
		SELECT *
		FROM Visit v
		WHERE v.pet_id = po.pet_id AND v.paid = FALSE
	)
)
```

```
MariaDB [bdale_cs355sp23]> SELECT * FROM vw_OwnersOutstandingPayments;
+----------+--------+-----------------+-----------------+
| Owner ID | Pet ID | Owner First Name | Owner Last Name |
+----------+--------+-----------------+-----------------+
|        1 |      1 | Selena           | Gomez           |
|        1 |     15 | Selena           | Gomez           |
|        3 |      3 | Mr.              | Worldwide       |
|        4 |      4 | Jonathon         | Thompson        |
|        4 |     18 | Jonathon         | Thompson        |
|        5 |      1 | Charles E        | Cheese          |
|        6 |      6 | Tobey            | Maguire         |
|        6 |     20 | Tobey            | Maguire         |
|        7 |      6 | Olzoneth         | Rilore          |
|        8 |      8 | Jack             | Skellington     |
|        9 |      9 | Barbara          | Matel           |
|        9 |     23 | Barbara          | Matel           |
|       12 |     12 | Scooby           | Doo             |
|       14 |      9 | Lizzie           | McGuire         |
|       15 |     28 | Allister         | Shinigami       |
+----------+--------+-----------------+-----------------+
15 rows in set (0.002 sec)
```

Procedure 1: sp_AddVisitService

This procedure safely adds an existing service to an existing visit. In this context, safely means that the service will only be added to this visit if the specified pet exists, the specified pet has an existing visit on the specified date, and this visit does not already have this service. In this, duplicate services are not allowed on a singular visit.

```
# Creating the procedure
DELIMITER $$

CREATE OR REPLACE PROCEDURE sp_AddVisitService
(
	IN petID_in INT UNSIGNED,
	IN date_in DATE,
	IN serviceName_in varchar(30)
)
BEGIN
	# Declare variables
	DECLARE CUSTOM_ERROR CONDITION FOR SQLSTATE '46000';
	DECLARE petExists BOOL DEFAULT FALSE;
	DECLARE visitExists BOOL DEFAULT FALSE;
	DECLARE visitHasNewService BOOL DEFAULT FALSE;

	# Make sure that the pet exists
	SET petExists = ( EXISTS (
		SELECT *
		FROM Pet p
		WHERE p.id = petID_in )
	);

	IF NOT petExists THEN
		SIGNAL CUSTOM_ERROR
			SET MESSAGE_TEXT = 'No pet found with that petID!';
	END IF;

	# Make sure that the visit exists
	SET visitExists = ( EXISTS (
		SELECT *
		FROM Visit v
		WHERE v.pet_id = petID_in AND v.`date` = date_in )
```

```
        );

        IF NOT visitExists THEN
                SIGNAL CUSTOM_ERROR
                        SET MESSAGE_TEXT = 'This pet does not have an existing visit on this
date!';
        END IF;

        # Make sure that this visit does not already have this service
        SET visitHasNewService = ( EXISTS (
                SELECT *
                FROM Visit_Service vs
                WHERE vs.pet_id = petID_in
                  AND vs.`date` = date_in
                  AND vs.service = serviceName_in
                )
        );

        IF visitHasNewService THEN
                SIGNAL CUSTOM_ERROR
                        SET MESSAGE_TEXT = 'Cannot add an duplicate service to a visit!';
        END IF;

        # Add the service to the visit
        INSERT INTO Visit_Service VALUES (petID_in, date_in, serviceName_in);

END $$

DELIMITER ;

# Example use of procedure
CALL sp_AddVisitService(28, '2020-10-10', 'Allergy testing');

# Checking (sql and before/after)
SELECT *
FROM Visit_Service vs
WHERE vs.pet_id = 28 AND vs.`date` = '2020-10-10';
```

sp_AddVisitService

Before

```
+--------+------------+---------------+
| pet_id | date       | service       |
+--------+------------+---------------+
|     28 | 2020-10-10 | Physical exam |
+--------+------------+---------------+
```

After

```
+--------+------------+-----------------+
| pet_id | date       | service         |
+--------+------------+-----------------+
|     28 | 2020-10-10 | Allergy testing |
|     28 | 2020-10-10 | Physical exam   |
+--------+------------+-----------------+
```

Procedure 2: sp_ApplySalaryChange

This procedure safely applies a pay increase/decrease to an existing employee. Safely means that the employee exists and that the new salary is greater than 0.

```
DELIMITER $$

CREATE OR REPLACE PROCEDURE sp_ApplySalaryChange
(
	IN employeeID_in INT UNSIGNED,
	IN newSalary numeric (12, 2)
)
BEGIN
	# Declare variables
	DECLARE CUSTOM_ERROR CONDITION FOR SQLSTATE '45000';
	DECLARE employeeExists BOOL DEFAULT FALSE;

	# Make sure the employee EXISTS
	SET employeeExists = ( EXISTS (
		SELECT *
		FROM Employee e
		WHERE e.id = employeeID_in )
	);

	IF NOT employeeExists THEN
		SIGNAL CUSTOM_ERROR
			SET MESSAGE_TEXT = 'No employee exists with this ID!';
	END IF;

	# Make sure salary is greater than 0
```

```
        IF newSalary <= 0 THEN
                SIGNAL CUSTOM_ERROR
                        SET MESSAGE_TEXT = 'The new salary is invalid -- New Salary must
be > 0!';
        END IF;

        # Apply the salary change
        UPDATE Employee
                SET salary = newSalary
                WHERE id = employeeID_in;

END $$

DELIMITER ;

# Example use of procedure
CALL sp_ApplySalaryChange(10, 75000.00);

# Checking (sql and before/after)
SELECT
        f_name AS 'First Name',
        l_name AS 'Last Name',
        Salary
FROM Employee e
WHERE e.id = 10;
```

sp_ApplySalaryChange

Before

```
+------------+-----------+----------+
| First Name | Last Name | Salary   |
+------------+-----------+----------+
| Elisa      | Capps     | 63000.00 |
+------------+-----------+----------+
```

After

```
+------------+-----------+----------+
| First Name | Last Name | Salary   |
+------------+-----------+----------+
| Elisa      | Capps     | 75000.00 |
+------------+-----------+----------+
```

Function 1: fn_VisitCostRemaining

Safely gets the cost remaining for an unpaid visit. In this context, safely means that the specified pet (of the visit) exists, that pet had a visit on the specified date, and that that visit has not already been paid for. Otherwise, it returns the total cost of the visit.

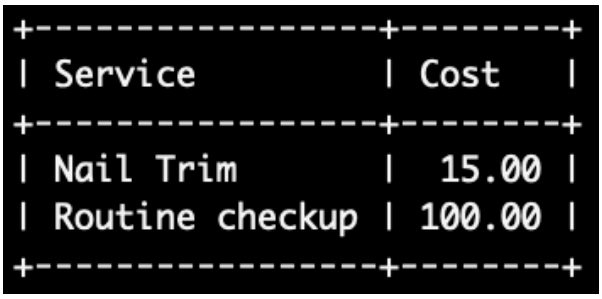
```
DELIMITER $$

CREATE OR REPLACE FUNCTION fn_VisitCostRemaining
(
	petID_in INT UNSIGNED,
	date_in Date
)
RETURNS NUMERIC(8, 2)
BEGIN
	# Declare variables
	DECLARE CUSTOM_ERROR CONDITION FOR SQLSTATE '45000';
	DECLARE petExists BOOL DEFAULT FALSE;
	DECLARE petVisitExists BOOL DEFAULT FALSE;
	DECLARE alreadyPaid BOOL DEFAULT FALSE;
	DECLARE costRem NUMERIC(8, 2);

	# Make sure that the pet exists
	SET petExists = ( EXISTS (
		SELECT *
		FROM Pet p
		WHERE p.id = petID_in )
	);

	IF NOT petExists THEN
		SIGNAL CUSTOM_ERROR
			SET MESSAGE_TEXT = 'No pet found with that petID!';
	END IF;

	# Make sure that the pet had a visit on that date
	SET petVisitExists = ( EXISTS (
		SELECT *
		FROM Visit v
		WHERE v.pet_id = pet_id AND v.`date` = date_in )
	);
```

```
	IF NOT petVisitExists THEN
		SIGNAL CUSTOM_ERROR
			SET MESSAGE_TEXT = 'Pet with this ID does not have a visit for that
date!';
	END IF;

	# Check that the visit has not already been paid for
	SET alreadyPaid = ( EXISTS (
		SELECT *
		FROM Visit v
		WHERE v.pet_id = petID_in
		  AND v.`date` = date_in
		  AND v.paid = TRUE )
	);

	IF alreadyPaid THEN
		SIGNAL CUSTOM_ERROR
			SET MESSAGE_TEXT = 'This visit has already been paid for!';
	END IF;

	# Calculate the cost of the visit
	SELECT SUM(s.cost)
	INTO costRem
	FROM Visit_Service vs JOIN Service s ON vs.service = s.name
	WHERE vs.pet_id = petID_in AND vs.`date` = date_in;

	RETURN costRem;

END; $$

DELIMITER ;


# Example use of procedure
SELECT fn_VisitCostRemaining(10, '2002-11-30');
```

```
# Checking above example use(sql and before/after)
SELECT
        vs.service AS Service,
        s.cost AS Cost
FROM Visit_Service vs JOIN Service s ON vs.service = s.name
WHERE vs.pet_id = 10
  AND vs.`date` = '2002-11-30';
```

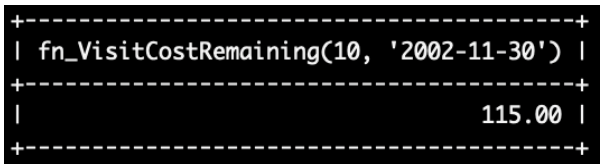
fn_VisitCostRemaining

Visit Services w/ Costs



Service	Cost
Nail Trim	15.00
Routine checkup	100.00

Function Output



fn_VisitCostRemaining(10, '2002-11-30')
115.00