Part C - PostgreSQL Implementation & Querying

Animal Shelter Management System

Big Data & Analytics

Ada Alkim Acikyol

Frenkli Zeqollari

Ipek Zobu

Mehlika Rana Akbay

PostgreSQL Schema Implementation

Below is the implementation of the PostgreSQL schema for the Animal Shelter Management System. The tables include necessary constraints such as primary keys, foreign keys, and checks.

1. CREATE TABLE SCRIPTS

```
CREATE TABLE Cage (
    cage_id VARCHAR(10) PRIMARY KEY,
    location VARCHAR(100) NOT NULL,
    capacity INT CHECK (capacity > 0),
    current_occupancy INT DEFAULT 0
);

CREATE TABLE Breed (
    breed_id VARCHAR(10) PRIMARY KEY,
    species VARCHAR(50) NOT NULL
);

CREATE TABLE Animal (
    animal_id VARCHAR(10) PRIMARY KEY,
    animal_name VARCHAR(100) NOT NULL,
    species VARCHAR(50) NOT NULL,
    age INT CHECK (age >= 0),
    gender VARCHAR(10),
    breed_id VARCHAR(10),
    arrival_date DATE NOT NULL,
    adoption_status VARCHAR(20) DEFAULT 'Available',
    cage_id VARCHAR(10)REFERENCES Cage (CageID)
);

CREATE TABLE Adopter (
    adopter_id VARCHAR(10) PRIMARY KEY,
    full_name VARCHAR(100) NOT NULL,
    email VARCHAR(100) UNIQUE NOT NULL,
    phone VARCHAR(20) UNIQUE NOT NULL,
    address VARCHAR(200) NOT NULL
);

CREATE TABLE Adoption (
    adoption_id VARCHAR(10) PRIMARY KEY,
    animal_id VARCHAR(10) REFERENCES Animal(animal_id),
    adopter_id VARCHAR(10) REFERENCES Adopter(adopter_id),
    adoption_date DATE NOT NULL,
    adoption_status VARCHAR(50) NOT NULL
);

CREATE TABLE Vaccination (
    vaccination_id VARCHAR(10) PRIMARY KEY,
    animal_id VARCHAR(10) REFERENCES Animal(animal_id),
```

```
    vaccination_type VARCHAR(100),
    vaccination_date DATE,
    next_due_vaccination DATE
);

CREATE TABLE MedicalRecord (
    record_id VARCHAR(10) PRIMARY KEY,
    animal_id VARCHAR(10) REFERENCES Animal(animal_id),
    visit_date DATE NOT NULL,
    diagnosis VARCHAR(200),
    treatment VARCHAR(200),
    vet_name VARCHAR(100) NOT NULL
);

CREATE TABLE Staff (
    staff_id VARCHAR(10) PRIMARY KEY,
    full_name VARCHAR(100) NOT NULL,
    role VARCHAR(50),
    phone VARCHAR(20),
    email VARCHAR(100) UNIQUE
);

CREATE TABLE Volunteer (
    volunteer_id VARCHAR(10) PRIMARY KEY,
    full_name VARCHAR(100) NOT NULL,
    email VARCHAR(100) UNIQUE,
    phone VARCHAR(20),
    availability VARCHAR(50)
);

CREATE TABLE Schedule (
    schedule_id VARCHAR(10) PRIMARY KEY,
    staff_id VARCHAR(10) REFERENCES Staff(staff_id),
    volunteer_id VARCHAR(10) REFERENCES Volunteer(volunteer_id),
    role_type VARCHAR(50) NOT NULL,
    shift_date DATE NOT NULL,
    start_time TIME NOT NULL,
    end_time TIME NOT NULL
);

CREATE TABLE Donation (
    donation_id VARCHAR(10) PRIMARY KEY,
    adopter_id VARCHAR(10) REFERENCES Adopter(adopter_id),
    donor_name VARCHAR(100) NOT NULL,
    amount DECIMAL(10,2) CHECK (amount > 0),
    date DATE,
    donor_type VARCHAR(50)
);

ALTER TABLE Animal
ADD CONSTRAINT fk_breed
FOREIGN KEY (breed_id) REFERENCES Breed (breed_id);
```

Sample Data Population

Description of Sample Data

To populate the PostgreSQL database, we used existing CSV files containing real-world-style mock data for all 11 tables in the schema:

- Animal.csv
- Adopter.csv
- Adoption.csv
- Breed.csv
- Cage.csv
- Donation.csv
- Medical Record.csv
- Schedule.csv
- Staff.csv
- Vaccination.csv
- Volunteer.csv

Each CSV file contains more than 50 rows of meaningful test data aligned with the schema's constraints, including proper primary key and foreign key references.

Sample Data Insertion

To populate the database with meaningful data, a total of 50+ rows were inserted using:
· Mockaroo.com to generate realistic test data.
· Data exported to CSV files.
· Uploaded using PostgreSQL's \copy command in psql.
This method ensured consistency and integrity while testing joins, constraints, and relationships.

Method of Import

All data was imported using the \copy command in psql, with the following format:
\copy tablename FROM '~/Downloads/file.csv' DELIMITER ';' CSV HEADER;

Sql:
\copy breed FROM '~/Downloads/Breed.csv' DELIMITER ';' CSV HEADER;
\copy cage FROM '~/Downloads/Cage.csv' DELIMITER ';' CSV HEADER;
\copy animal FROM '~/Downloads/Animal.csv' DELIMITER ';' CSV HEADER;
\copy adopter FROM '~/Downloads/Adopter.csv' DELIMITER ';' CSV HEADER;

\copy adoption FROM '~/Downloads/Adoption.csv' DELIMITER ';' CSV HEADER;
\copy donation FROM '~/Downloads/Donation.csv' DELIMITER ';' CSV HEADER;
\copy "medical record" FROM '~/Downloads/MedicalRecord.csv' DELIMITER ';' CSV HEADER;
\copy schedule FROM '~/Downloads/Schedule.csv' DELIMITER ';' CSV HEADER;
\copy staff FROM '~/Downloads/Staff.csv' DELIMITER ';' CSV HEADER;
\copy vaccination FROM '~/Downloads/Vaccination.csv' DELIMITER ';' CSV HEADER;
\copy volunteer FROM '~/Downloads/Volunteer.csv' DELIMITER ';' CSV HEADER;

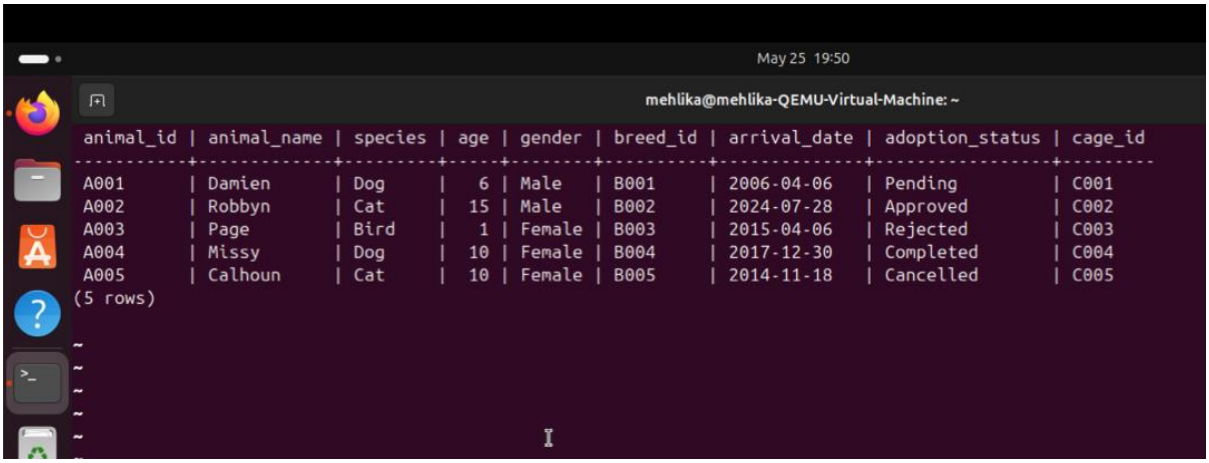
The delimiter was ; because all CSVs use semicolons instead of commas.

Important Note: Due to foreign key dependencies, the import order followed this structure:

1. Breed
2. Cage
3. Animal
4. Adopter
5. Staff
6. Volunteer
7. Adoption
8. Vaccination
9. Medical Record
10. Schedule
11. Donation

Successful Import Example

- SELECT * FROM animal LIMIT 5;



Data Cleaning Notes

- CSV headers were matched exactly to SQL table column names
- All date formats were normalized to YYYY-MM-DD
- Extra carriage returns and encoding issues were removed with sed
- Foreign key violations were tested and resolved prior to final imports

SQL QUERY SECTION

This section demonstrates five meaningful SQL queries, each highlighting a different concept. The queries are relevant to the animal shelter's business logic and are supported with brief explanations and results.

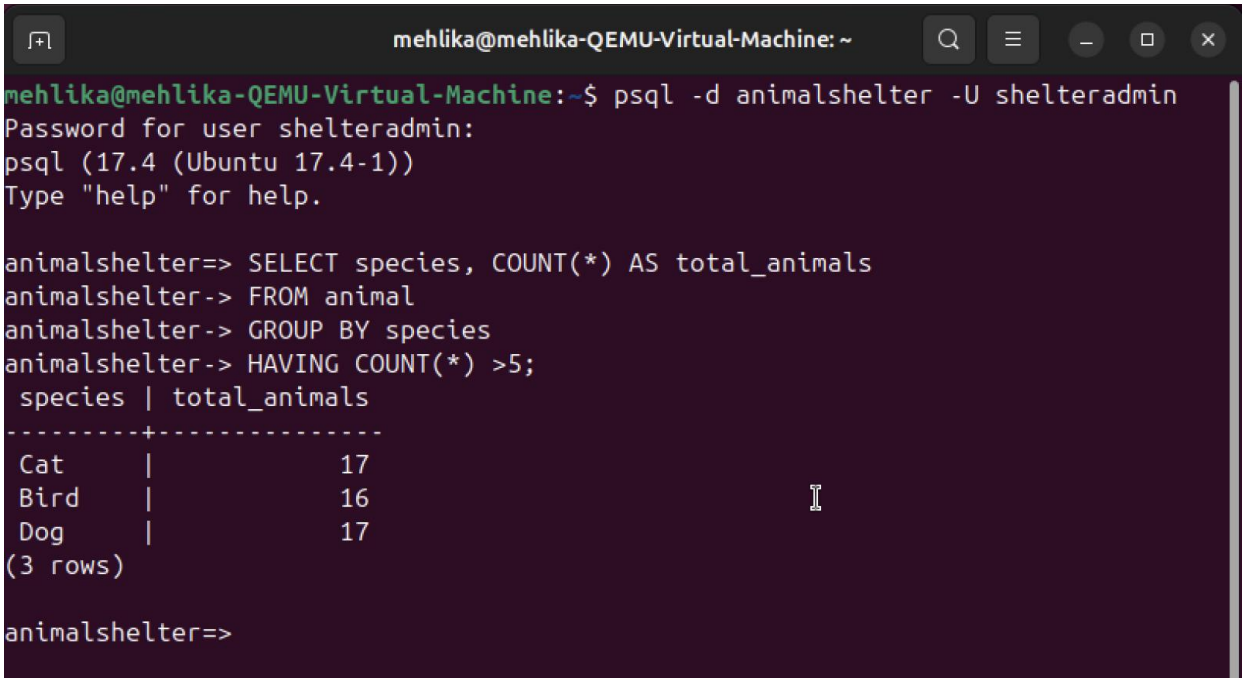
Query 1:
Aggregation with GROUP BY and HAVING

SQL Code:

```
SELECT species, COUNT(*) AS total_animals
FROM animal
GROUP BY species
HAVING COUNT(*) > 5;
```

Explanation:
This query counts how many animals belong to each species. It only returns species with more than 5 animals. This helps staff focus on the most common species in the shelter.

Result:



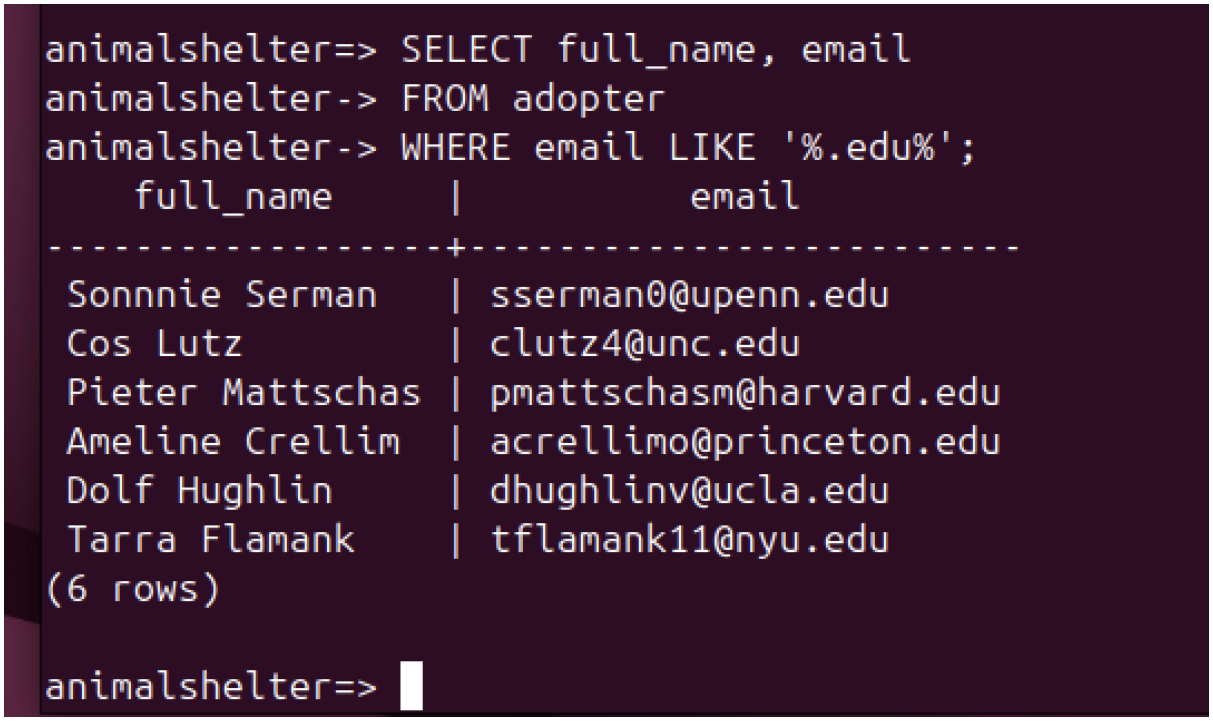
Query 2:
Filtering with WHERE and LIKE

SQL Code:

```
SELECT full_name, email
FROM adopter
WHERE email LIKE '%.edu%';
```

Explanation:
This query filters adopters whose email addresses contain ".edu", identifying those affiliated with academic institutions. This can support outreach or donation campaigns.

Result:



Query 3:
ORDER BY (Top Donors)

SQL Code:

```
SELECT donor_name, amount
FROM donation
ORDER BY amount DESC
LIMIT 5;
```

Explanation:
This query displays the top 5 donors who contributed the most. It is helpful for recognizing major contributors and encouraging continued support.

Result:

```
animalshelter=> SELECT donor_name, amount
animalshelter-> FROM donation
animalshelter-> ORDER BY amount DESC
animalshelter-> LIMIT 5;
    donor_name    | amount
------------------+---------
 Gabriel Minnette | 1800.00
 Tarra Flamank    | 1500.00
 Holly Autry      | 1500.00
 Torrie Bolus     | 1500.00
 Leola Granger    | 1200.00
(5 rows)

animalshelter=>
```

Query 4:

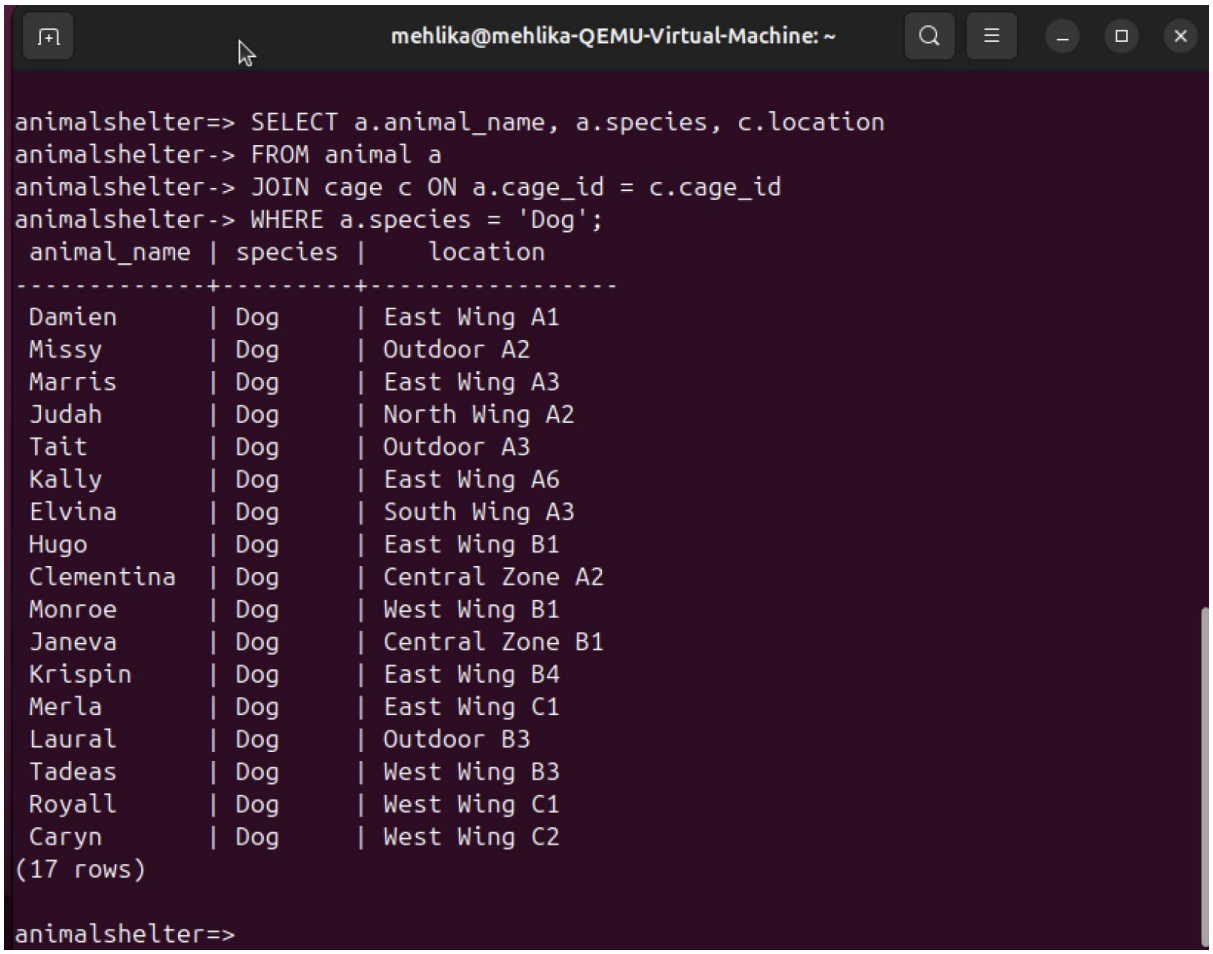
JOIN with Filtering

SQL Code:

```
SELECT a.animal_name, a.species, c.location
FROM animal a
JOIN cage c ON a.cage_id = c.cage_id
WHERE a.species = 'Dog';
```

Explanation:
This query joins animal and cage tables to show all dogs and their cage locations. It aids in space planning and identifying dog housing within the shelter.

Result:



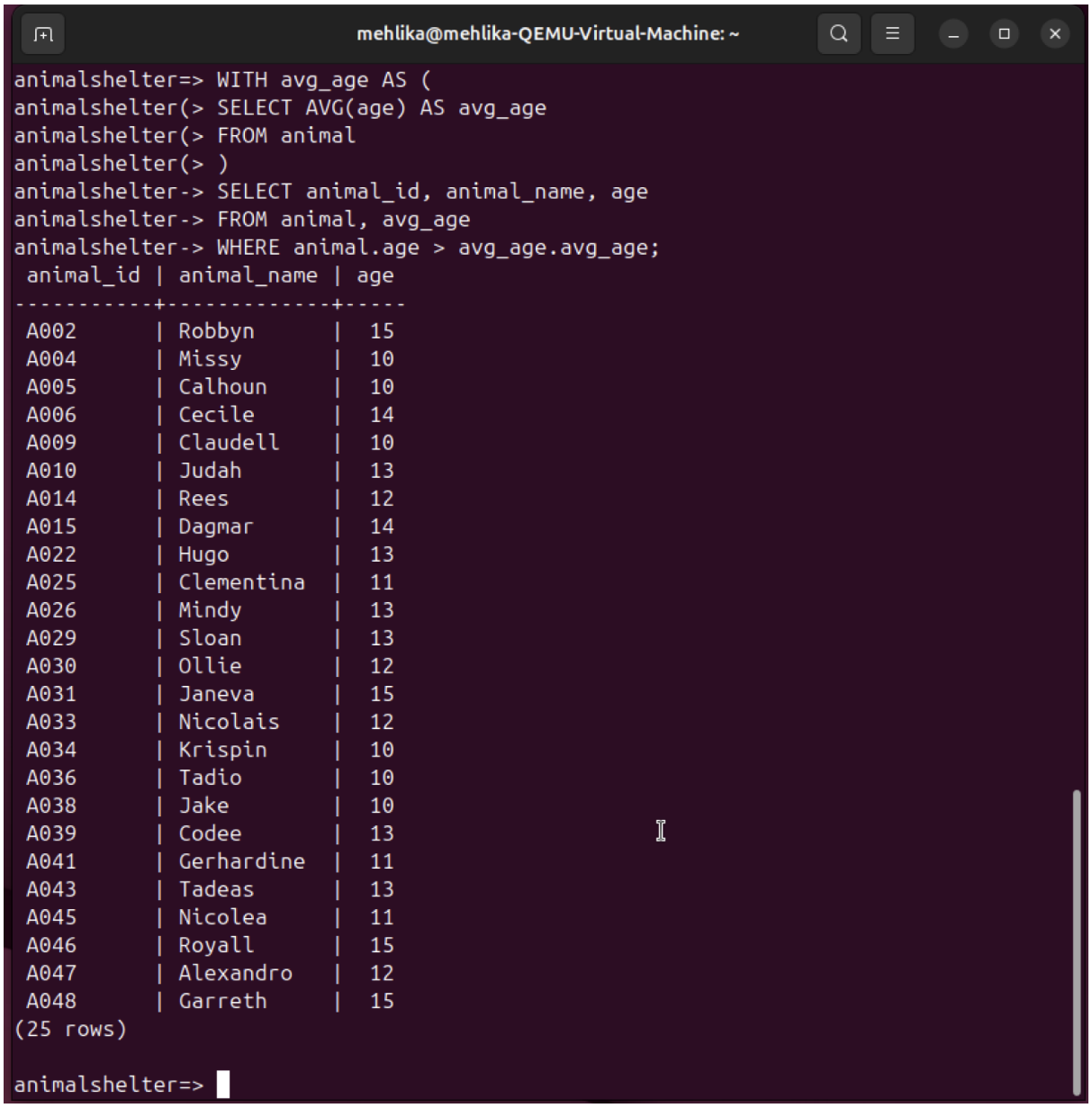
Query 5:
CTE (Common Table Expression)

SQL Code:

```
WITH avg_age AS (
   SELECT AVG(age) AS avg_age
   FROM animal
)
SELECT animal_id, animal_name, age
FROM animal, avg_age
WHERE animal.age > avg_age.avg_age;
```

Explanation:
This query identifies animals older than the average age using a Common Table Expression (CTE). It is useful to detect older animals who might need prioritized attention or care.

Result:



Challenges Faced & Solutions

During the implementation and testing phases of this PostgreSQL database project, we faced several complex and critical challenges. These issues were resolved through a combination of documentation review, error-driven debugging, and adherence to best practice. Below we outlined the primary challenges encountered and the effective strategies employed to overcome them.

1. Managing Foreign Key Dependencies During Import

Challenge:
Initial attempts to populate the database resulted in foreign key constraint violations, particularly in tables such as animal, adoption, and medicalrecord. This was due to importing dependent tables before their referenced tables.

Solution:
A detailed import strategy was developed to ensure referential integrity. Tables were loaded in dependency order: parent tables like breed and cage were imported first, followed by dependent tables. This sequencing resolved all constraint violations without modifying schema integrity.

2.Inconsistent CSV Formatting and Delimiter Errors

Challenge:
The default delimiter in PostgreSQL's \copy command is a comma, whereas the provided CSV files used semicolons. This mismatch initially caused parsing failures.

Solution:
We explicitly defined the delimiter as ; during import by updating all \copy commands to include DELIMITER ';'. Additionally, all CSV files were validated to ensure proper quoting, line endings, and header-row alignment with the SQL schema.

3. Post-Schema Adjustment for Referential Integrity

Challenge:
The breed_id field in the animal table was initially implemented as a regular column. Upon further normalization analysis, it became clear that this should be a foreign key referencing the breed table.

Solution:
To correct this without dropping the table and risking data loss, an ALTER TABLE statement was used to add the foreign key constraint post hoc:

```
ALTER TABLE Animal
ADD CONSTRAINT fk_breed
FOREIGN KEY (breed_id) REFERENCES Breed(breed_id);
```

This approach maintained schema integrity and aligned with 2NF principles without affecting existing data.

4.Lack of Access to Files During Import via UTM Ubuntu

Challenge:

The macOS host system used UTM to emulate Ubuntu. Initial attempts to import CSV files using \copy failed due to the inability to access macOS files from the virtual machine.

Solution:
We resolved this by transferring the files directly inside the Ubuntu VM using WhatsApp Web, and then confirmed the download path with pwd before running the import commands.

These challenges illustrate a typical data engineering workflow and reinforced the importance of best practices in schema design, ensuring data consistency, and understanding PostgreSQL's operational logic.

Final Formatting & Submission

To ensure a successful and complete submission, the following structure and formatting guidelines were applied:

Report Structure

The final report is compiled into a single PDF document and includes the following clearly labeled sections:

1. **Schema Implementation**
    a. Full CREATE TABLE SQL scripts for all 11 tables
    b. Screenshots showing successful table creation and structure confirmation
2. **Sample Data Population**
    a. Description of data generation and import method
    b. Explanation of delimiter choice and file path handling
    c. Screenshots showing successful imports and sample data queries
3. **SQL Query Section**
    a. Five diverse SQL queries covering aggregation, filtering, ordering, joins, and CTEs
    b. Each query is accompanied by a brief explanation and a screenshot of the result
4. **Challenges & Solutions**
    a. Documentation of technical issues encountered
    b. Step-by-step description of how each problem was identified and solved
5. **Final Submission Check**
    a. Report is clean, professional, and logically formatted

b. All code blocks use monospace font
c. Screenshots are sharp, annotated where needed, and relevant to each step
d. All CSV files used have been validated and aligned with schema columns