A. Initial Phase

- Identify at least three distinct user types. For each user type, describe the specific data they need to store, retrieve, and update in the database.
  Three user types could be:
  Shelter staff: they need full access to animal records, adopter information, adoption history, event schedules, volunteer assignments, and donation records. They can add/update animal profiles, process adoptions, schedule medical treatments, manage events, and record donations.

  Adopters: they need access to the available animals for adoption, adoption requirements, application status, and upcoming adoption events. They should be able to submit adoption applications, update personal information, view animal profiles, and track application status.

  Vets: animal medical records, vaccination history, treatment plans

- Explain how these needs influence the structure of the database

  The different users must constrain our database schema by determining how we organize and connect information. Shelter staff need complete access to all information on animals, so our database will need to maintain detailed records with a straightforward link between tables. Adopters must see only adoptable animals, determining how we handle animal status. Veterinarians care about medical information, requiring detailed health records connected to each animal. These varied demands lead us to create separate tables for animals, adopters, adoptions, medical records, and staff and define established relationships between them.

B. Conceptual Design Phase

Describe how you would represent the pet adoption database using the relational model and any advantages to this approach. Include at least four tables with their attributes, primary keys, foreign keys and relationships. Do NOT reuse tables from the midterm exam.

Animals Table
Attributes: animal_id (PK), name, species, breed, age, gender
Relationships:
"Receives" (One-to-many) with Medical_Records (one animal can receive many medical treatments)
"Participates" (One-to-one) with Adoptions (one animal can participate in at most one adoption)

Adopters Table

Attributes: adopter_id (PK), first_name, last_name, phone, approval_status
Relationships:
"Completes" (One-to-many) with Adoptions (one adopter can complete many adoptions)

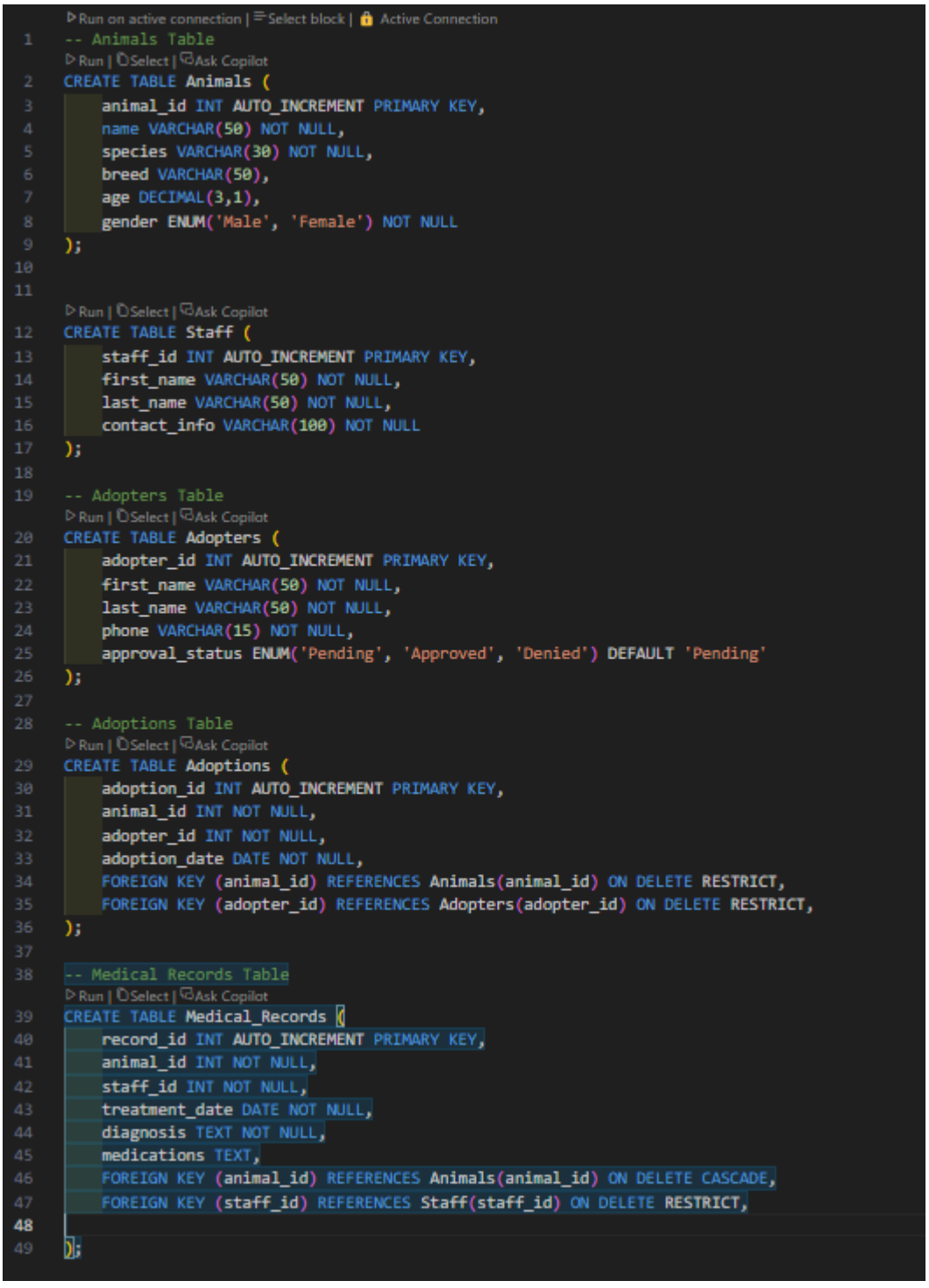
Adoptions Table
Attributes: adoption_id (PK), animal_id (FK), adopter_id (FK), adoption_date
Relationships:
"Includes" (Many-to-one) with Animals (many adoptions can include one animal)
"Processed_By" (Many-to-one) with Adopters (many adoptions can be processed by one adopter)

Medical_Records Table
Attributes: record_id (PK), animal_id (FK), staff_id (FK), treatment_date, diagnosis, medications
Relationships:
"Documents" (Many-to-one) with Animals (many medical records can document one animal's health)
"Created_By" (Many-to-one) with Staff (many medical records can be created by one staff member)

Staff Table
Attributes: staff_id (PK), first_name, last_name, contact_info
Relationships:
"Provides" (One-to-many) with Medical_Records (one staff member can provide many medical treatments)

C. Logical-Design Phase

- Convert your conceptual schema into an SQL-based relational schema. Write SQL commands to define tables that specify the following:
    - Primary keys
    - Foreign keys
    - Relationships
    - Constraints (i.e., NOT NULL, UNIQUE, CHECK, etc.)



```sql
-- Animals Table
CREATE TABLE Animals (
    animal_id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(50) NOT NULL,
    species VARCHAR(30) NOT NULL,
    breed VARCHAR(50),
    age DECIMAL(3,1),
    gender ENUM('Male', 'Female') NOT NULL
);


CREATE TABLE Staff (
    staff_id INT AUTO_INCREMENT PRIMARY KEY,
    first_name VARCHAR(50) NOT NULL,
    last_name VARCHAR(50) NOT NULL,
    contact_info VARCHAR(100) NOT NULL
);

-- Adopters Table
CREATE TABLE Adopters (
    adopter_id INT AUTO_INCREMENT PRIMARY KEY,
    first_name VARCHAR(50) NOT NULL,
    last_name VARCHAR(50) NOT NULL,
    phone VARCHAR(15) NOT NULL,
    approval_status ENUM('Pending', 'Approved', 'Denied') DEFAULT 'Pending'
);

-- Adoptions Table
CREATE TABLE Adoptions (
    adoption_id INT AUTO_INCREMENT PRIMARY KEY,
    animal_id INT NOT NULL,
    adopter_id INT NOT NULL,
    adoption_date DATE NOT NULL,
    FOREIGN KEY (animal_id) REFERENCES Animals(animal_id) ON DELETE RESTRICT,
    FOREIGN KEY (adopter_id) REFERENCES Adopters(adopter_id) ON DELETE RESTRICT,
);

-- Medical Records Table
CREATE TABLE Medical_Records (
    record_id INT AUTO_INCREMENT PRIMARY KEY,
    animal_id INT NOT NULL,
    staff_id INT NOT NULL,
    treatment_date DATE NOT NULL,
    diagnosis TEXT NOT NULL,
    medications TEXT,
    FOREIGN KEY (animal_id) REFERENCES Animals(animal_id) ON DELETE CASCADE,
    FOREIGN KEY (staff_id) REFERENCES Staff(staff_id) ON DELETE RESTRICT,

);
```

Part II: Database Users & Administrators

A pet adoption database system serves a variety of users, each with unique roles, responsibilities, and requirements. Understanding these different user types helps ensure that the system is designed to meet their needs effectively.

1. Describe the responsibilities and interaction a naïve user would have with the pet adoption database system. Include an example of this type of user interacting with the system. Do NOT reuse any examples from the midterm exam.

   A naïve user is responsible for basic data entry, simple searches, and generating standard reports. Naïve users interact with the system through graphical interfaces. They enter information about new animals, update existing records when animals receive medical care, and process adoption paperwork. They search the database using criteria like animal type, age, etc. an example of this user interacting with the system is a front desk volunteer uses the system to register a new animal arrival at the shelter

2. Describe the responsibilities and interaction an application programmer would have with the pet adoption database system. Include an example of this type of user interacting with the system. Do NOT reuse any examples from the midterm exam.

   Application programmers design user interfaces, write code for data processing, implement security measures, and troubleshoot technical issues. For interactions, Application programmers interact with the database by writing code that generates and executes SQL statements. an example of this user interacting with the system is a programmer creating a feature that allows potential adopters to search for compatible animals based on certain criteria.

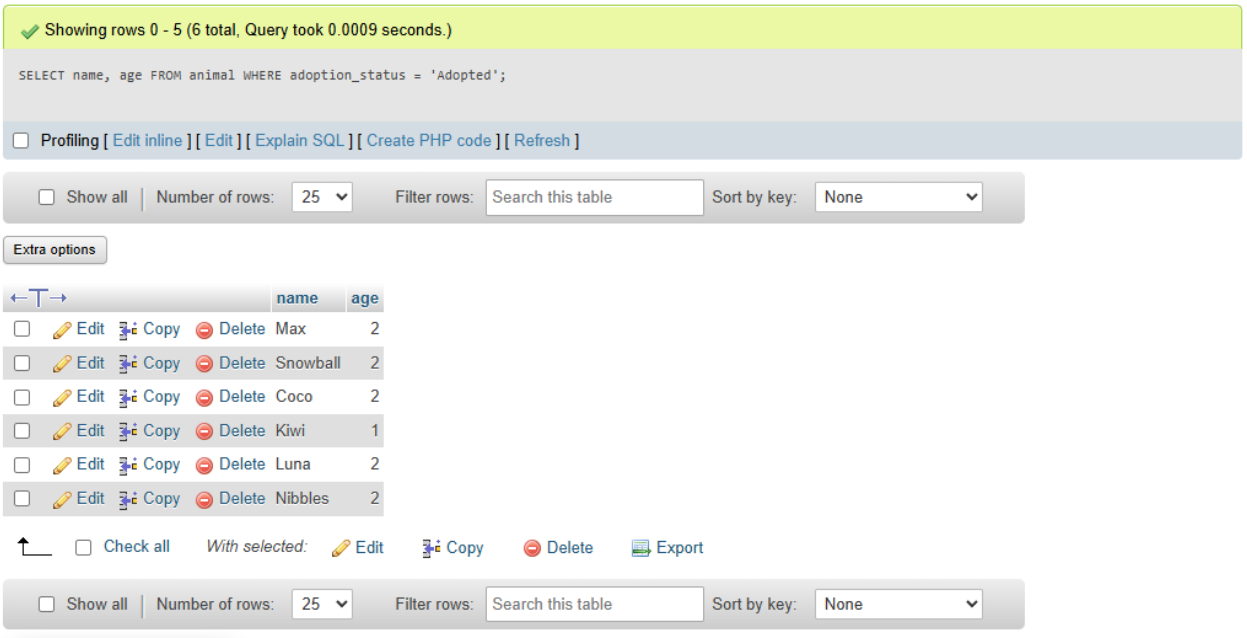
3. Describe the responsibilities and interaction a sophisticated user would have with the pet adoption database system. Include an example of this type of user interacting with the system. Do NOT reuse any examples from the midterm exam.

   Sophisticated users can write custom SQL queries and create reports that aren't available through standard interfaces. Sophisticated users interact with the database through query tools or administrative interfaces that allow direct SQL access. an example of this user interacting with the system is a shelter manager querying the database to analyze seasonal adoption trends across different animal types and ages.

4. Describe the responsibilities and interaction a database administrator would have with the pet adoption database system. Include an example of this type of user interacting with the system. Do NOT reuse any examples from the midterm exam.
The Database Administrator manages the technical aspects of the database system. Database administrators interact with the system at the most technical level, working directly with the database management system compared to the application which is handled by the Application programmer. an example of this user interacting with the system is a DBA monitoring the system performance during peak hours, and implementing indexes to improve response time for all users.

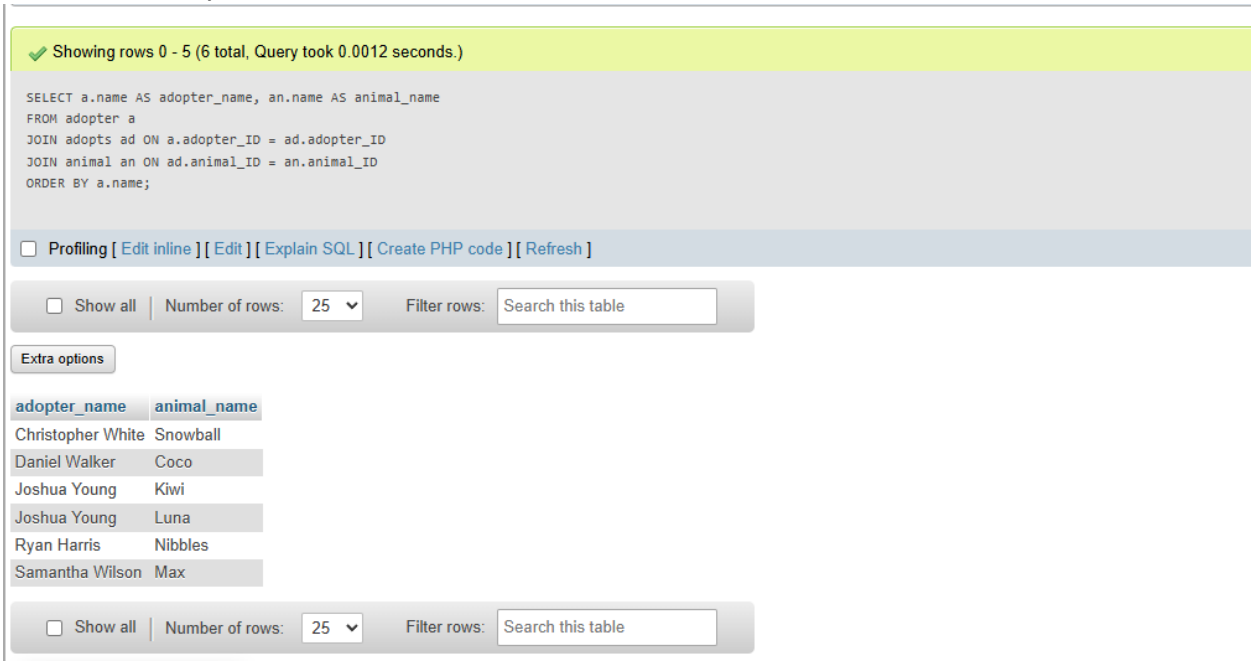
Part IV: Relational Algebra & SQL:

Find the names and ages of all animals that have been adopted.
Relational algebra: π name, age (σ adoption_status = 'Adopted' (animal))
SQL: SELECT name, age FROM animal WHERE adoption_status = 'Adopted';



Find the names of each adopter and the name(s) of the animals they adopted.
Relational Algebra: π adopter.name, animal.name (adopter ⋈ adopter_ID=adopter_ID adopts ⋈ animal_ID=animal_ID animal)
SQL:
SELECT adopter.name AS adopter_name, animal.name AS animal_name
FROM adopter, adopts, animal

WHERE adopter.adopter_ID = adopts.adopter_ID
AND adopts.animal_ID = animal.animal_ID
ORDER BY adopter.name;



Showing rows 0 - 5 (6 total, Query took 0.0012 seconds.)

```
SELECT a.name AS adopter_name, an.name AS animal_name
FROM adopter a
JOIN adopts ad ON a.adopter_ID = ad.adopter_ID
JOIN animal an ON ad.animal_ID = an.animal_ID
ORDER BY a.name;
```

Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]

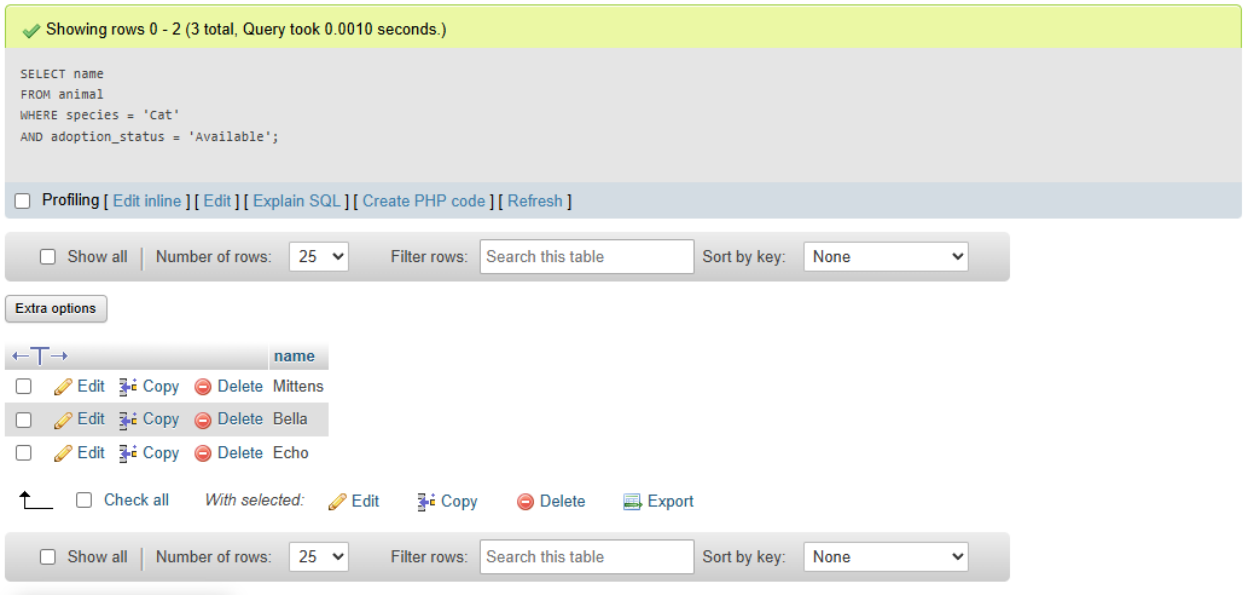
Show all | Number of rows: 25 Filter rows: Search this table

Extra options

adopter_name	animal_name
Christopher White	Snowball
Daniel Walker	Coco
Joshua Young	Kiwi
Joshua Young	Luna
Ryan Harris	Nibbles
Samantha Wilson	Max

Show all | Number of rows: 25 Filter rows: Search this table

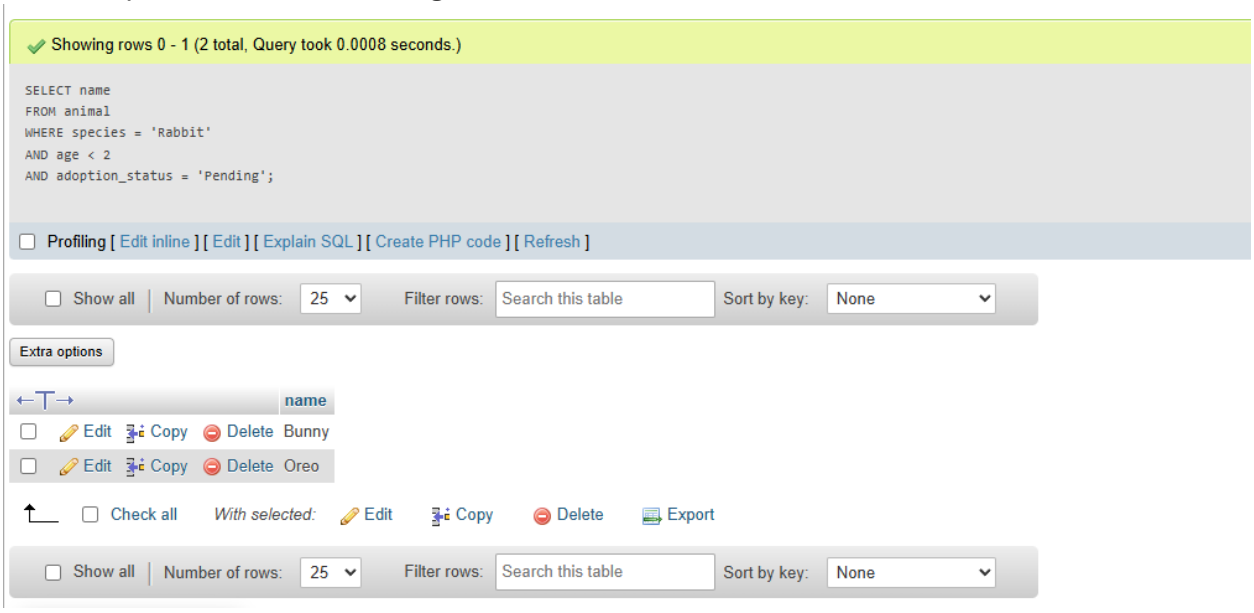
Find the names of male cats, female cats or both who are available for adoption.
Relational algebra: π name (σ species = 'Cat' AND adoption_status = 'Available' (animal))
SQL: SELECT name
FROM animal
WHERE species = 'Cat'
AND adoption_status = 'Available';



Find the names of all rabbits that are both less than 2 years old and have a pending adoption.

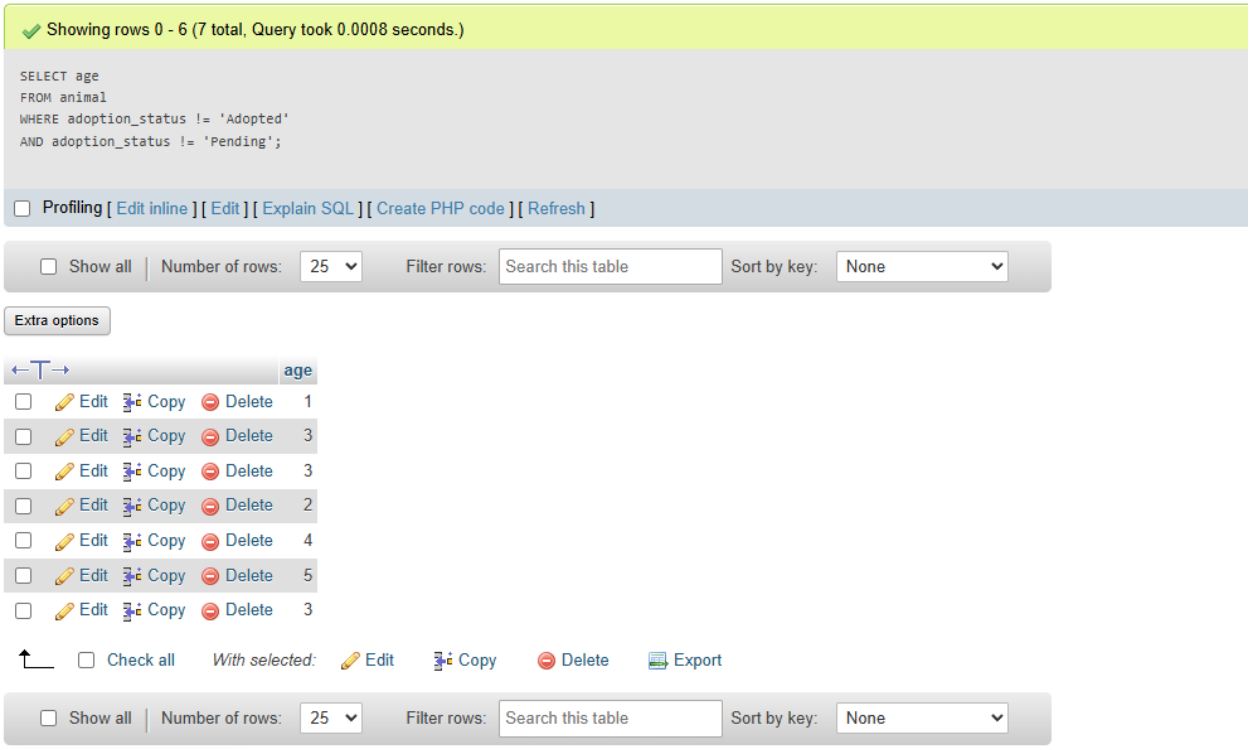
Relational Algebra: π name (σ species = 'Rabbit' AND age < 2 AND adoption_status = 'Pending' (animal))

SQL: SELECT name
FROM animal
WHERE species = 'Rabbit'
AND age < 2
AND adoption_status = 'Pending';



Find the ages of all animals who have not been adopted nor a pending adoption.

Relational Algebra: π age (σ adoption_status ≠ 'Adopted' AND adoption_status ≠ 'Pending' (animal))
SQL: SELECT age
FROM animal
WHERE adoption_status != 'Adopted'
AND adoption_status != 'Pending';



Part V: Database Design Pitfalls:

1. Bark Twain calls out sick. We remove his shift date and time with the following command:

DELETE FROM volunteers **WHERE** name = 'Bark Twain';

What anomaly arises with this SQL query? Explain how this issue affects the database.

This creates a deletion anomaly. We lose all information about Bark Twain, including his contact details and assignment. His information must be re-entered completely if he returns.

2. Clawdia Pawsworth changed her email to clawdia.paws@gmail.com to be more secure. We update her email with the following command:
UPDATE volunteers
SET contact_info = '(555) 369-8520, clawdia.paws@gmail.com'
WHERE name = 'Clawdia Pawsworth' **AND** shift_date = '2025-03-02';

What anomaly arises with this SQL query? Explain how this issue affects the database.

This creates an update anomaly. The update only changes her email for the March 2nd shift, leaving inconsistent contact information across her records.

3. Fetch Waggington is a new volunteer, but hasn't been given an assignment or shift yet. We add him with the following command:
INSERT INTO volunteers (name, assignment, contact_info, shift_date, shift_time)
VALUES ('Fetch Waggington', , '(555) 222-3333, fetch.w@barkmail.com', ,);

What anomaly arises with this SQL query? Explain how this issue affects the database.

This creates an insertion anomaly. The SQL fails because we can't add a volunteer without assignment and shift information, forcing us to either use fake data or wait until scheduling is finished.

Part VI: Database Normalization

A. First Normal Form (1NF)

- Identify and describe all violations of 1NF. Provide examples from the table that demonstrate each violation.
  The volunteers table has violations including multi-valued attributes, non-atomic values, inconsistent formats, and no primary key.
  Example: 'Bark Twain', 'Buddy, Walk', '(555) 987-4321, bark.twain@woofmail.com', '2025-03-01', '11-12 PM' contains comma-separated values in the assignment and contact_info fields.

- Rewrite the table to remove all 1NF violations. Include your fully compliant 1NF table.

Volunteers_1nf:

Field	Type	Null	Key	Default	Extra
volunteer_id	int(11)	NO	PRI	*NULL*	auto_increment
name	varchar(100)	YES		*NULL*	
phone	varchar(20)	YES		*NULL*	
email	varchar(100)	YES		*NULL*	
animal_name	varchar(50)	YES		*NULL*	
task	varchar(50)	YES		*NULL*	
shift_date	date	YES		*NULL*	
shift_time	varchar(20)	YES		*NULL*	

B. Second Normal Form (2NF)

- Identify and describe all violations of 2NF. Provide examples from the table that demonstrate each violation.
  The volunteers_1NF table has violations because volunteer information is repeated across multiple records.
  Example: Clawdia Pawsworth's name, phone '(555) 369-8520', and email 'clawdia.paws@kittenmail.com' appears in two separate rows - one for her March 2nd shift with Tweety and another for her March 3rd shift with Thumper.

- Rewrite the table to remove all 2NF violations. Include your fully compliant 2NF table.
  Volunteers_2nf:

Field	Type	Null	Key	Default	Extra
volunteer_id	int(11)	NO	PRI	*NULL*	auto_increment
name	varchar(100)	YES		*NULL*	
phone	varchar(20)	YES		*NULL*	
email	varchar(100)	YES		*NULL*	

Assignments_2nf:

Field	Type	Null	Key	Default	Extra
assignment_id	int(11)	NO	PRI	*NULL*	auto_increment
volunteer_id	int(11)	YES	MUL	*NULL*	
animal_name	varchar(50)	YES		*NULL*	
task	varchar(50)	YES		*NULL*	
shift_date	date	YES		*NULL*	
shift_time	varchar(20)	YES		*NULL*	

C. Third Normal Form (3NF)

- Identify and describe all violations of 3NF. Provide examples from the table that demonstrate each violation.

  After creating the 2NF tables (volunteers_2NF and assignments_2NF), the assignments_2NF table still contains 3NF violations in the form of transitive dependencies:

  Task information depends on the task name, not directly on the primary key. example, the task "Walk" appears in multiple records (for Bark Twain with Buddy and for Droolius Caesar with Rex).

  Animal information is stored as names rather than referencing the existing animal table. example, "Buddy" appears as text instead of using a foreign key to the animal table, creating a dependency on animal names rather than the primary key.

  Rewrite the table to remove all 3NF violations. Include your fully compliant 3NF table.

  volunteers_3NF:

Field	Type	Null	Key	Default	Extra
volunteer_id	int(11)	NO	PRI	*NULL*	auto_increment
name	varchar(100)	YES		*NULL*	
phone	varchar(20)	YES		*NULL*	
email	varchar(100)	YES		*NULL*	

  Tasks_3nf:

Field	Type	Null	Key	Default	Extra
task_id	int(11)	NO	PRI	*NULL*	auto_increment
task_name	varchar(50)	YES		*NULL*	

  shifts_3NF:

Field	Type	Null	Key	Default	Extra
shift_id	int(11)	NO	PRI	*NULL*	auto_increment
volunteer_id	int(11)	YES	MUL	*NULL*	
animal_id	int(11)	YES	MUL	*NULL*	
task_id	int(11)	YES	MUL	*NULL*	
shift_date	date	YES		*NULL*	
shift_time	varchar(20)	YES		*NULL*	