PET ADOPTION PORTAL

Team Name: Detonators

Abhijeet Sugam
asugam
asugam@buffalo.edu

Kushan Devarajegowda
kushande
kushande@buffalo.edu

Victor Vats
victorku
victorku@buffalo.edu

I. PROBLEM STATEMENT

The problem statement tries to build data models for pet adoption platforms. It is challenging for pet adoption centers to maintain track of the pets available for adoption, their breed's features, and vital stats.

Choosing a breed or type of pet when someone plans to adopt one is tricky. Following their decision, they find it challenging to locate a pet adoption center with the breed of dog they want. Many factors must be considered before choosing a suitable breed or type of pet.

The bottom line is that, for people, it is hard to find the different types of available pets/breeds and their characteristics and decide on adopting one. It is hard for adoption centers to maintain records of the pets available for adoption. As well as the breed characteristics and the owner's records.

The reason for choosing Databases over Excel files in the Project is due to the following reasons:

- "Data integrity" in the tables ensures that the value entered in a field is of the specified data type (e.g., you can't enter a text into a numeric field) or doesn't exceed the specified length.
- "Referential integrity" in the relationships between tables ensures the referenced value in another table exists.
- Compared to an Excel spreadsheet, a database is, by design, more suitable to handle large amounts of data and allows querying the results efficiently.
  , Unlike Excel, our database system can be accessed by multiple users concurrently.
- Using a Database system gives us privileges in managing secure access.
- Complex relationships can be accommodated with great ease and simplistic representation.

II. PROPOSED SOLUTION

Our love for pets and the hassles around their adoption intrigued us to envisage the idea of creating this Pet Database A system that addresses the problem by adopting the following key points:

- Making all the breed data about the pet and their characteristics, pictures, and much more data available in one place
- Providing a portal for pet adoption with all the preference filters based on the availability of the pets and storing the details like owner info, adoption date, etc.
- Employing a machine learning algorithm to recommend the most suitable pet for adoption based on inputs from users' preferences.
- Gathering all the data about pets (currently limited to dogs. However, it could be scaled) and support Searching breeds, finding a suitable match, and filtering based on the preferences/characteristics.

Our goal is to make the adoption process hassle-free.

III. TARGET USERS

Users:

- Every pet lover would love to explore the new breeds of pets that are currently available.
- Every pet adoption center wants to make its adoption process hassle-free.
- Every individual looking to adopt a new pet would use this data to find recommendations.
- Every individual who wants to report an abandoned pet to provide shelter for it.
- Every individual who wants to promote their pets for adoption.

Administers:

- Adoption center employees.

Database Admin:

- The platform's owner built the platform's infrastructure so users can access it. They will keep the database updated and ensure everyone can use the system.

Some real-life use cases where it eases down hassles:

- Pet adoption centers that manually manage their adoption process can use our system to make it quick and easy.
- It can be used to perform much research to come up with decisions to make a final adoption.
- Reduces the burden of running from one pet shelter to another in search of a specific breed and set of traits for the person looking for adoption.
- A person looking to adopt a particular breed of a pet can check on pictures present in the gallery to know better.
- Users can check the availability and unavailability of the dog in the pet shelter.
- The characteristics users want in a pet can be checked and researched based on their needs, such as adaptability, friendliness, trainability, etc.

IV. SCHEMA DESCRIPTION

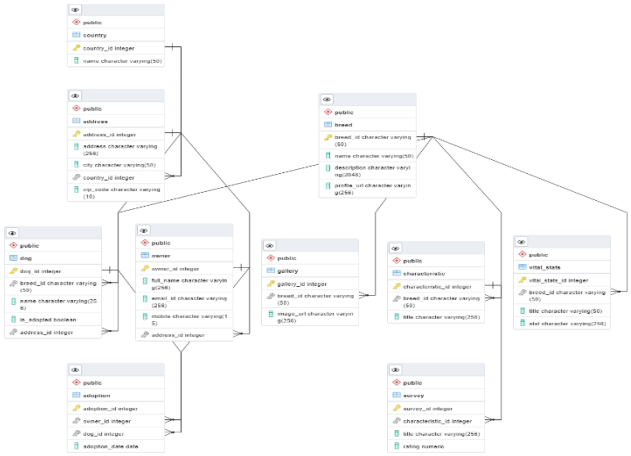

Figure 1: ERD Diagram

- **Breed**: As seen in the schema, we have a 'Breed' table, which has 'breed_id' as the primary key to identify each breed of dog uniquely.
- **Characteristic**: This table has 'characteristic_id' assigned to identify each character pet has uniquely. A breed can have multiple of these characteristics mapped in this table. So, we have used 'breed_id' as a foreign key in this table, uniquely identifying breed in table "breed."
- **Survey**: This table uses 'survey_id' to uniquely identify the surveys performed and is the primary key of this table. Surveys are related to how a dog is rated on some scale based on a particular characteristic of any breed. In this table, we also have 'characteristic_id,' a foreign key. So multiple survey_id is mapped to a specific characteristic_id of a species.
- **Gallery**: This table has 'gallery_id' as the primary key to uniquely map multiple image URLs to a breed. Hence, 'breed_id' is a foreign key for this table, referring to the 'breed' table.
- **Vital_stat**: This table stores unique signs of vitals using 'vital_stat_id'; hence it is the primary key for this table. It uses 'breed_id' as a foreign key to refer to the 'breed table.'
- **Owner**: This table uses 'owner_id' to uniquely identify each owner's entry in the table; the adoption table will use this column to establish a relation.
- **Dog:** This table 'dog_id' uniquely identifies each dog (not breed) available for adoption. In addition, it uses 'breed_id ' as a foreign key to retrieve the breed of the dog.
- **Adoption**: This table uses 'adoption_id' to uniquely store each adoption entry. Hence, it is the primary key for the table. The table uses 'dog_id' and 'owner_id' as foreign keys to refer to necessary details from the 'dog' and 'owner' tables.
- **Country**: This table uses 'country_id' to uniquely store a set of countries available. This acts as a primary key for this table and maps to address the table acting as a foreign key.
- **Address**: This table uses 'address_id' to uniquely store addresses, thus acting as the primary key for this table. It also employs 'country_id' from the 'Country' table as a foreign key to establish the relationship between the two relations.

V. ATTRIBUTES DESCRIPTION

Notes:

- We have added the data type and the maximum length of the string in square braces; e.g., Varchar[50] means a maximum of 50 characters.
- We have added the foreign tag key (FK) after the attribute name to indicate it. All the FK follows the DELETE CASCADE policy.
- We have added the tag NOT NULL(NN) after the attributes if it cannot be null.

Breed:

- *Breed_Id* (Varchar[50], NN): Stores the uniquely assigned Id to each breed of pet. Hence this attribute is fit to be the primary key.
- *Name* (Varchar[50], NN): Stores the name of the Breed.
- *Description* (Varchar[2048], NN): Stores the details related to the breed in terms of an overview of what is unique about the breed and any trivia related to that breed.
- *Profile_url* (Varchar[256], NN): Stores the URL of the website from where one can get more details related to that dog breed.

Characteristic:

- *Characteristic_id*(Integer, NN): Stores the assigned unique id of a dog that has been mapped to some characteristic. This being unique makes it suitable for the primary key.
- *Breed_id*(Varchar[50], NN): This is the primary key in the Breed table, which is referred to here to establish a relationship between those two tables. Hence, it's a foreign key in this table.
- *Title*(Varchar[256], NN)*:* This stores the few-word textual representation of the various characters of the dog.

Survey:

- *Survey_id*(Integer, NN) integer, not null default nextval: Stores the assigned unique id of surveys mapped to surveys done regarding the nature of dogs. This being unique makes it suitable for the primary key.
- *Characteristic_id*(Integer, NN) integer not null: Stores the assigned unique id of a dog that has been mapped to some characteristic. This primary key in the characteristic table is used as a foreign key to reference the characteristic table.
- *Title*(Varchar[256], NN): Stores the textual description of characteristic_id using a few words to give a good idea of the pet's behavior.
- *Rating*(Integer, NN): Stores the rating(1-5) in terms of how much a particular characteristic of a pet has based on the surveys done on that pet category.

Gallery:

- *Gallery_id*(Integer, NN): Stores the id to map the gallery corresponding to each dog's images uniquely. Acts as the primary key in this table.

- *Breed_id*(Varchar[50], NN): Stores the uniquely assigned Id to each dog breed. As this attribute is the primary key in the breed table, so hence being referenced here as a foreign key to establish a relation between the two tables.
- *Image_url*(Varchar[256], NN): Stores the sample image URL of each pet present in the table, which can be used to view by the user.

Vital_stats:
- *Vital_stats_id*(Integer, NN): Stores the ids to a unique set of Vitals used to characterize a breed. A breed can have multiple groups of vital_status_id associated with it. As it is unique, hence can be treated as the primary key for this table.
- *Breed_id*(Varchar[50], NN): Stores the uniquely assigned Id to each dog breed. As this attribute is the primary key in the breed table, so hence being referenced here as a foreign key to establish the relation between the two tables.
- *Title*(Varchar[50], NN): Stores the textual one-worded description assigned to the vital status of a breed. These can be their weight, height, average age, etc.
- *Stat*(Varchar[256], NN): Stores values for each vital stat such as the value of their weight, height, average age, etc.

Owner:
- *Owner_id*(Integer, NN): Stores the unique id assigned to each owner to be populated in this table. This is the primary key for this table.
- *Full_name*(Varchar[256], NN): Stores the owner's full name.
- *Email_id*(Varchar[256], NN): Stores the email ids of the owners
- *Mobile*(Varchar[15], NN): Stores the mobile numbers of the owners.
- *AddressId*(Integer, NN): Stores the ids of the address of the owners present in the owner table.

Dog:
- *Dog_id*(Integer, NN): Stores id assigned to each dog (not breed) which is required to put an identification to the dog available for adoption.
- *Breed_id*(Varchar[50], NN): Stores the uniquely assigned Id to each dog breed. As this attribute is the primary key in the breed table, so hence being referenced here as a foreign key to establish a relation between the two tables.
- *Name*(Varchar[256], NN): Stores the dog's name available for adoption.
- *Is_adopted*(Boolean, NN): Stores the status of whether a dog is available for adoption using true or false values.
- *Address_Id*(Integer, NN): Stores the ids of the address of the owners present in the owner table to achieve the normalized form of the table.

Country:
- *CountryId* (Integer, NN): Stores the IDs of the countries present in the address table to establish the mapping and helps us normalize the table.
- *Name* (Varchar[50], NN): Stores the countries' names in the address table.

Adoption:
- *Adoption_id*(Integer, NN): Stores id assigned to each adoption which took the place of a dog (not breed) required to put an identification to dog adoption. It acts as the primary key in this table.
- *Owner_id*(Integer, NN): Stores the unique id assigned to each owner to be populated in this table. This is the primary key in the owner table, referenced here as a foreign key.
- *Dog_id*(Integer, NN): Stores id assigned to each dog (not breed) which is required to put an identification to the dog available for adoption.
- *Adoption_date*(Date, NN): Stores the dates on which the adoption was made in date format.

Address:
- *Address_Id*(Integer, NN): Stores the IDs of the addresses provided as part of the owner relation. It was created as part of normalization to keep the change of address hassle separate.
- *Address*(Varchar[256], NN): Stores the address itself as part of the mapping to the address_id.
- *City*(Varchar[50], NN): Stores the owner's city provided as part of their address.
- *Country_id*(Integer, NN): Stores the corresponding IDs of the owner's country provided as part of their address.
- *Zip_code*(Varchar[10], NN): Stores the owner's zip provided as part of their address.

VI. FDs AND BCNF OF TABLE

Table 1: Functional dependencies

Table Name	Functional Dependencies
Breed	breed_id→name, description, profile_url profile_url→breed_id, name, description
Characteristic	characteristic_id→breed_id, title
Gallery	gallery_id →breed_id, image_url
Vital_Stat	vital_stats_id →breed_id, title, stat
Survey	survey_id→characteristic_id, title, rating
Dog	dog_id→breed_id,name,is_adopted,address_id
Address	address_id→address,city,country_id, zip_code
Country	country_id →name
Owner	owner_id→full_name,email_id,mobile,address_id email_id→full_name,owner_id,mobile,address_id
Adoption	adoption_id →owner_id, dog_id, adoption_date

Table 2: BCNF Proof

Table name	Candidate Keys	Prime attributes
breed	breed_id, profile_url	breed_id, profile_url
characteristic	characteristic_id	characteristic_id
gallery	gallery_id	gallery_id
vital_stats	vital_stats_id	vital_stats_id
survey	survey_ id	survey_ id
dog	dog_id	dog_id
address	address_id	address_id
country	country_id	country_id
owner	owner_id, email_id	owner_id, email_id
adoption	adoption_id	adoption_id

From the above-given table, one can see that the FDs in each relation adhere to two given conditions to be consistent with the BCNF form:

- They are non-trivial
- The attributes on the left side of FDs are super keys.

VII. SCHEMA ENHANCEMENT

- We have decomposed the address details from the owner table and reused it to store the address of the dog listed for adoption., also we have added a new table country to keep the country by its ID to save memory.
- We have added the attributes 'is_adopted' in the adoption table to keep track of the adoption status of the dog to avoid joining with the adoption table to minimize query execution time.
- We have identified the standard length of the string in all the tables and revised accordingly. For example, for the name attribute, we changed the size from Varchar (256) to Varchar (50).

VIII. BASIC QUERIES

```
--1. Add new dog for adoption listing.

INSERT INTO Dog
            (breed_id,
             NAME,
             address_id)
VALUES      ('RatTerrier',
             'Lenord',
             361)
--2. Add new address

INSERT INTO Address
            (address,
             city,
             country_id,
             zip_code)
VALUES      ('3288 Main St, Buffalo',
             'New York',
             (SELECT country_id
              FROM   Country
              WHERE  NAME = 'United States'),
             '14214')
--3. Update mobile number of a owner given email id.

UPDATE owner
SET    mobile ='17162564670'
WHERE  email_id ilike 'susan.wilson@puppyworld.in'


--4. Update adoption status of all the dogs in the dog table.

UPDATE Dog
SET    is_adopted = true
WHERE  dog_id IN (SELECT dog_id
                  FROM   Adoption)

--5. Delete owner with given email id

DELETE owner
WHERE  email_id ilike 'susan.wilson@puppyworld.in'
```

Figure 2: Basic queries

IX. ADVANCED QUERIES

```
--1. SELECT THE DOGS COUNT BY BREED THAT ARE AVAILABLE FOR THE ADOPTION.


SELECT breed_id AS Breed,
       Count(*) AS AvailableCount
FROM   Dog
WHERE  dog_id NOT IN (SELECT dog_id
                      FROM   Adoption)
GROUP  BY breed_id

--2. Select name of the dog breeds which are not kids friendly (i.e has <2 rating out of 5)

SELECT B.NAME,
       S.title,
       S.rating
FROM   Breed B,
       Characteristic C,
       Survey S
WHERE  B.breed_id = C.breed_id
       AND C.characteristic_id = S.characteristic_id
       AND S.title = 'Kid-Friendly'
       AND S.rating < 2

--3. Select dog breeds that are neither easy to train nor easy to groom.

SELECT B.NAME,
       S.title,
       S.rating
FROM   Breed B,
       Characteristic C,
       Survey S
WHERE  B.breed_id = C.breed_id
       AND C.characteristic_id = S.characteristic_id
       AND S.title IN ( 'Easy To Groom', 'Easy To Train' )
       AND S.rating = 1

--4. Select all the owners from India and list their adopted dog_name, breed_name,
--   and adoption_date and order by adoption date.

SELECT O.full_name AS owner,
       D.NAME      AS dog_name,
       B.NAME      AS breed_name,
       A.adoption_date
FROM   Owner O
       INNER JOIN Adoption A
               ON O.owner_id = A.owner_id
       INNER JOIN Dog D
               ON D.dog_id = A.dog_id
       INNER JOIN Breed B
               ON B.breed_id = D.breed_id
       INNER JOIN Address Ad
               ON O.address_id = Ad.address_id
       INNER JOIN Country C
               ON Ad.country_id = C.country_id
WHERE  C.NAME = 'India'
ORDER  BY A.adoption_date


--5. Rank the country names by their adoption rate.

SELECT DISTINCT( C.name ),
               Rank ()
                 OVER (
                   ORDER BY Count(A.adoption_id) DESC) rank,
               Count(A.adoption_id)                    AS total_adoption
FROM   Owner O,
       Adoption A,
       Dog D,
       Breed B,
       Address Ad,
       Country C
WHERE  O.owner_id = A.owner_id
       AND A.dog_id = D.dog_id
       AND O.address_id = Ad.address_id
       AND Ad.country_id= C.country_id
       AND D.breed_id = B.breed_id
GROUP  BY C.name
HAVING Count(A.adoption_id) >= 3
ORDER  BY rank

--6. Delete all the adoption records that are older than 10 years

CREATE OR REPLACE PROCEDURE DeleteAdoptionsOlderThanFiveYears()
LANGUAGE SQL
BEGIN ATOMIC
  DELETE FROM adoption
    WHERE  adoption_date < NOW()-interval '5 year';

    DELETE FROM dog
    WHERE  dog_id NOT IN (SELECT dog_id
                          FROM   adoption)
           AND is_adopted = true;
END;
```

Figure 3: Advanced queries

X. QUERY ANALYSIS AND OPTIMIZATIONS

We have employed two of the following query optimization techniques:

- Subquery & Join
- Indexing

Use case 1: Rank country names by their adoption rate.

Before Optimization: Query uses a subquery to fetch data from another table, fetching cross products of the two table records to search for results.

Execution Time: 15850 ms

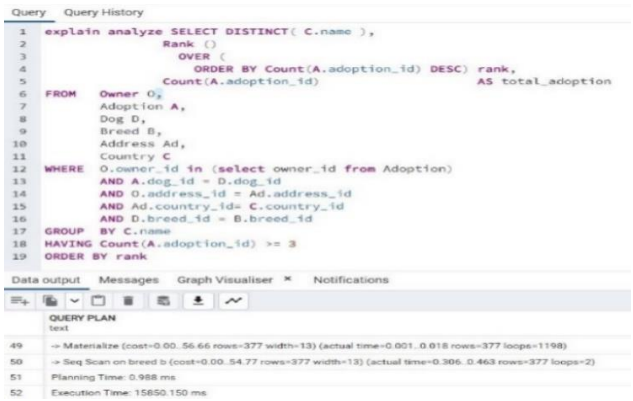

Figure 4: Use case 1 query before optimization

After Optimization: Query uses an inner join to establish a mapping between 2 tables, reducing the size of temporary results on the join operation, leading to merging two table records and ignoring duplicate records.

Execution Time: 119 ms

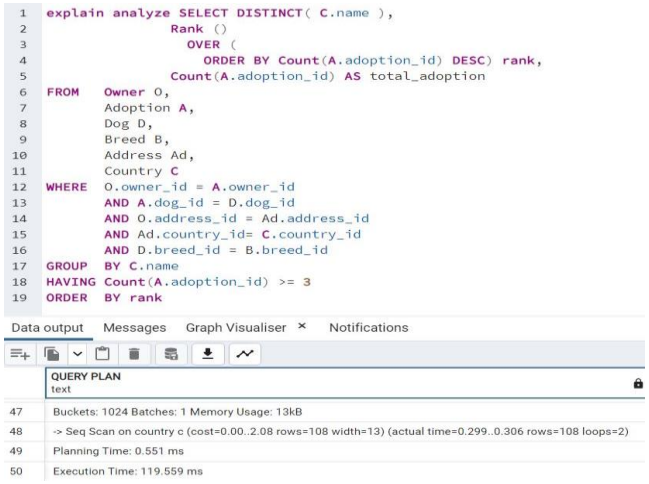

Figure 5: Use case 1 query after optimization

Conclusion: By replacing the subquery with the join mechanism, we reduced execution time by **133** times.

Use case 2: Search by dog name adopted between given dates.

Before Optimization: Query uses a subquery to fetch data from another table, leading to fetch the cross products of the two table records to search for results.

Execution Time: 13581 ms

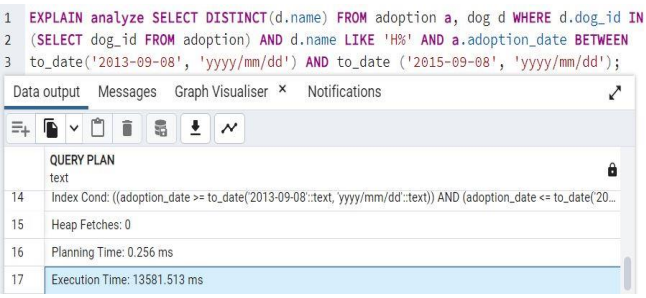

Figure 6: Use case 2 query before optimization

After Optimization (Using Join): Query uses an inner join to establish a mapping between 2 tables, reducing the size of temporary results on the join operation, leading to merging two table records and ignoring duplicate records.

Execution Time: 15 ms

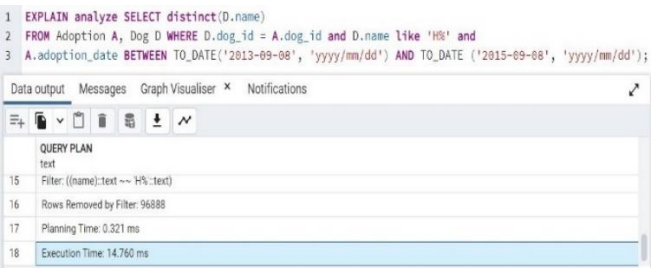

Figure 7: Use case 2 query after subquery optimization

After Optimization (Using Indexing): Query uses the few most accessed attributes, and due to sequential disk index seek, fetching records takes more time. Hence, we decided to index the attribute that is most frequently fetched,
i.e., dog_name and adoption_date.

Execution Time: 7 ms

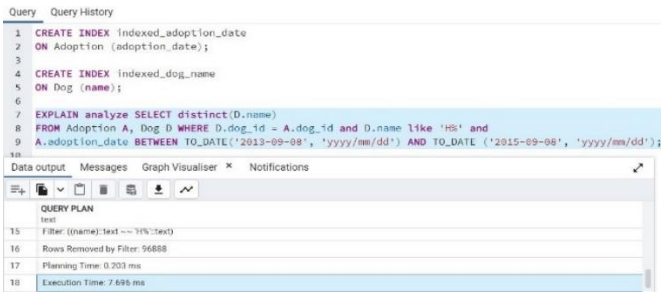

Figure 8: Use case 2 query after index optimization

Conclusion: Firstly, using join reduced execution time by **905** times; subsequently, Indexing was employed to reduce it further by **130** times.

Use case 3: Check the adoption status of a dog using the breed id, name, and address of the dog.

Before Optimization: Without indexing, all the attributes are fetched using a sequential scan; since this is a direct match query, a sequential scan is unsuitable.

Execution Time: 6.209 ms

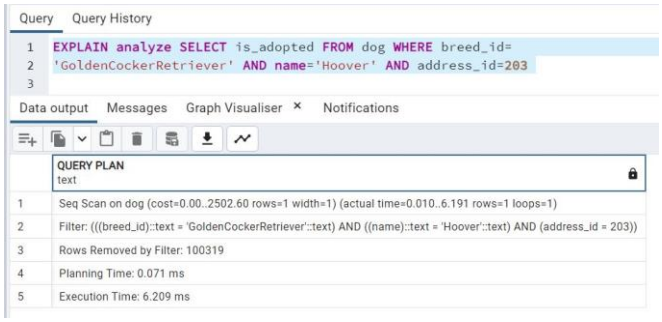

Figure 9: Use case 3 query before optimization

After Optimization: We have indexed all three attributes, breed_id, address_id & dog_name, to reduce the disk scan, resulting in faster access. Following the execution plan attached below, we can observe that all the attributes are fetched using the indexing we created.

Execution Time:0.103 ms

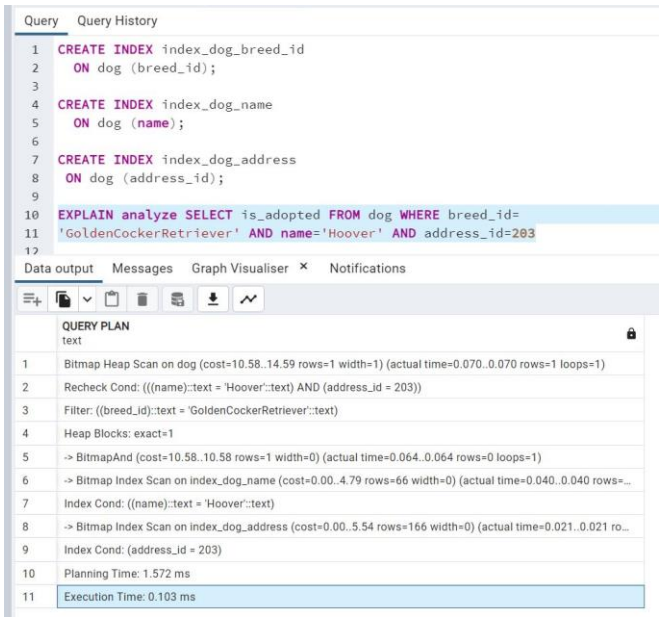

Figure 10: Use case 3 query after optimization

Conclusion: Hence by employing indexing, we reduced execution time by **60** times by reducing the access time of the indexed attributes of the query.

XI. APPLICATION DEVELOPMENT

Pet adoption assistant is a chatbot application that can help the adoption center admin quickly access the data from the database in the natural English language.

When a user makes any query, we use NLP to identify the nature/intent of the query and then fetch and execute the query from the database and respond to the question back to the user in human-readable text format.

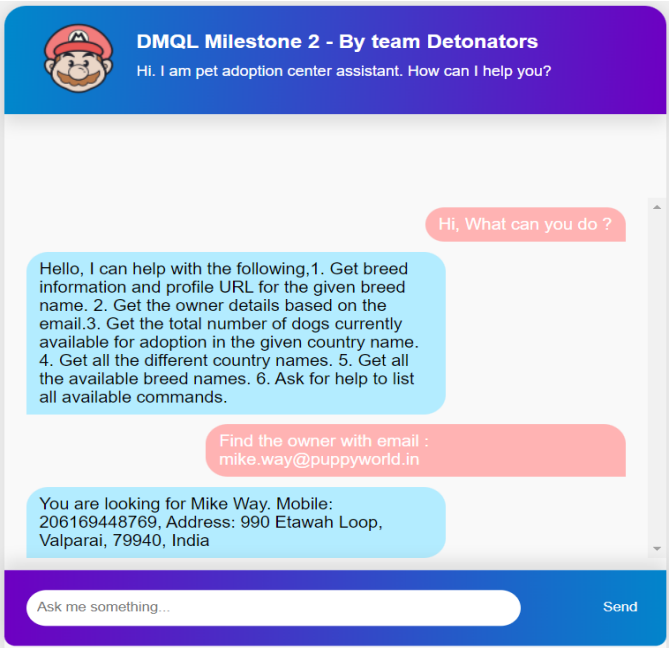

Figure 11: Application greeting

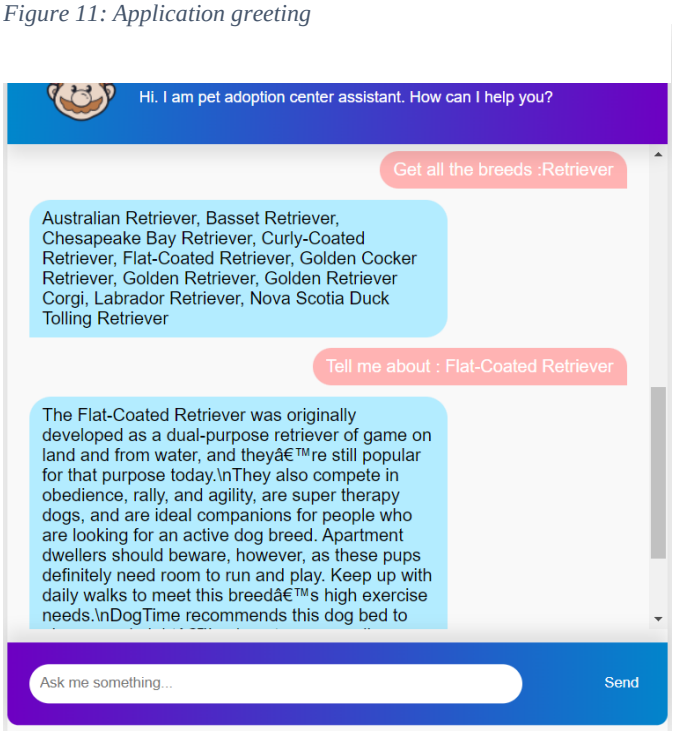

Figure 12: Application user interaction

The following are a few supported queries users can make:

- Retrieve breed information by name.
- Retrieve all the breed names with matching keywords.
- Retrieve the pet owner's information.
- Retrieve the number of pets available for adoption in the given country.
- Retrieve all the country names.

Back-End: Python, Flask, NLP, Pytorch, and PostgreSQL
Front-End: HTML, CSS, JavaScript

Conclusion: Although the chatbot serves fewer queries at this moment. However, the scope of these queries can be extended in the future based on the requirements; hence, our chatbot can handle questions of the more significant score as developed.

XII. CHALLENGES ENCOUNTERED:

The below points highlight significant challenges we encountered during various phases of our project and how we tackled those:

1. Data Gathering and ingestion:
Mocking the data was difficult as it would have no significant meaning to our project. Hence, we collected real-world data by developing a web scraper from scratch, which gathered thousands of rows of records to be fed to our database from the website Dogtime.com.
We performed data cleaning to get rid of junk or corrupt data. Also, we have created. SQL scripts generate the schema as per our requirement, and then we use load.sql for each table to push data into each table. Additionally, we have a readme.txt file that walks you through performing the actions mentioned above.

2. Optimization Issue:
Another issue we noticed was query execution time with searches including numerous joins, cartesian products of tables using up to 100K tuples, and other complicated queries; obtaining a fraction of data from such relations proved expensive. We performed an analysis of these unoptimized queries. We identified the problematic questions that were degrading database performance and developed an improved version of such queries by introducing indexing of attributes, subqueries, optimizing joins, and other processing that reduces the tuple size due to join operations improving query time.

3. Data Integrity Issue:
Atomic deletion from multiple tables was challenging; for instance, removing older entries from the adoption table also should delete corresponding records in the dog table. We have solved this using a stored procedure that ensures the atomic deletion from multiple tables by maintaining data integrity.

XIII. CONTRIBUTIONS:

There are no individual contributors to the project; we believe all of us have put in equal effort to achieve the result. However, we did share responsibility equally to ensure the consistently paced progress of our project.

Below is the responsibility segregation of individuals:

Abhijeet contributed to Application front-end development, connecting the database to the backend and query analysis and use-case development.

Kushan worked majorly on Application API and backend development, NLP classification, and data training used in the query classification and, Performing FDs and BCNF validations.

Victor contributed to Developing a data scraping application, data cleanup, insertion, and all the supporting load, creating SQL file bundling.

Finally, Schema design, Reporting, Advanced queries and optimizations, and every other detail in the report are the few critical tasks performed in a group that cannot be assigned to an individual and done in a group with thorough discussions involved.

XIV. REFERENCES:

- [1] https://dogtime.com/
- [2] https://www.officetuto.com/
- [3] https://www.postgresql.org/
- [4] University Database from dvd-rental