DTSC-691

Capstone Database Final Project

2023SU1

By: Matt Dalgetty

Table of Contents

Background

Tree companies are customer-facing service organizations. They often have a small team of managers and sales representatives, and a large number of laborers which actually perform the work. In these organizations there can be a disconnect between management and workers. The work is performed outside in different locations and management remains in the office. This can create negative outcomes for customers and decrease the overall quality of work performed. Another problem these companies face is technological illiteracy. Due to the blue-collar nature of the industry, often both management and workforce are not technologically skilled. The workers will have no ability to interact with the data necessary to make “best practice” work decisions. And management will end up performing repetitive IT tasks which could be easily automated.

This disconnect between management and workers is a problem seen in many industries. What is unique to the arboriculture industry is that workers are performing treatments on living organisms which are essential to the environment, trees. Many of the treatments performed on trees can only safely be done every few years. Some of the trees worked on are very old with complex histories, and quite valuable both to the customers and to the environment for the services they provide. Therefore it is of the utmost importance that the job not only gets done, but maintains the integrity of the tree it is being performed on. It is essential that workers and management have access to the history of work that has been performed on individual trees. In order to accomplish this I will be creating a database that employees will be able to easily interact with both in the office and on the job site.

Problem Objective

The objective of this project was to create a relational database for a small tree company and build a webapp that performs operations on it. The database needed to be hosted on a cloud server in order for it to be accessible by the webapp. The webapp needed to be hosted in order for workers to have access to it in the field. The database would be managed in a postgreSQL client by managers to clean and organize data collected by the workers.

All of this will be created from the ground up. Due to the accelerated 7 week time frame, a fictional tree company will be created for the sake of this project. The database will be filled with a small number of example entries. And the webapp will be hosted until grading is completed.

This document serves as a complete report for the project. It will contain all SQL components in the DDL and DML sections. All Python and HTML code for the webapp will be attached at the time of submission.

Software

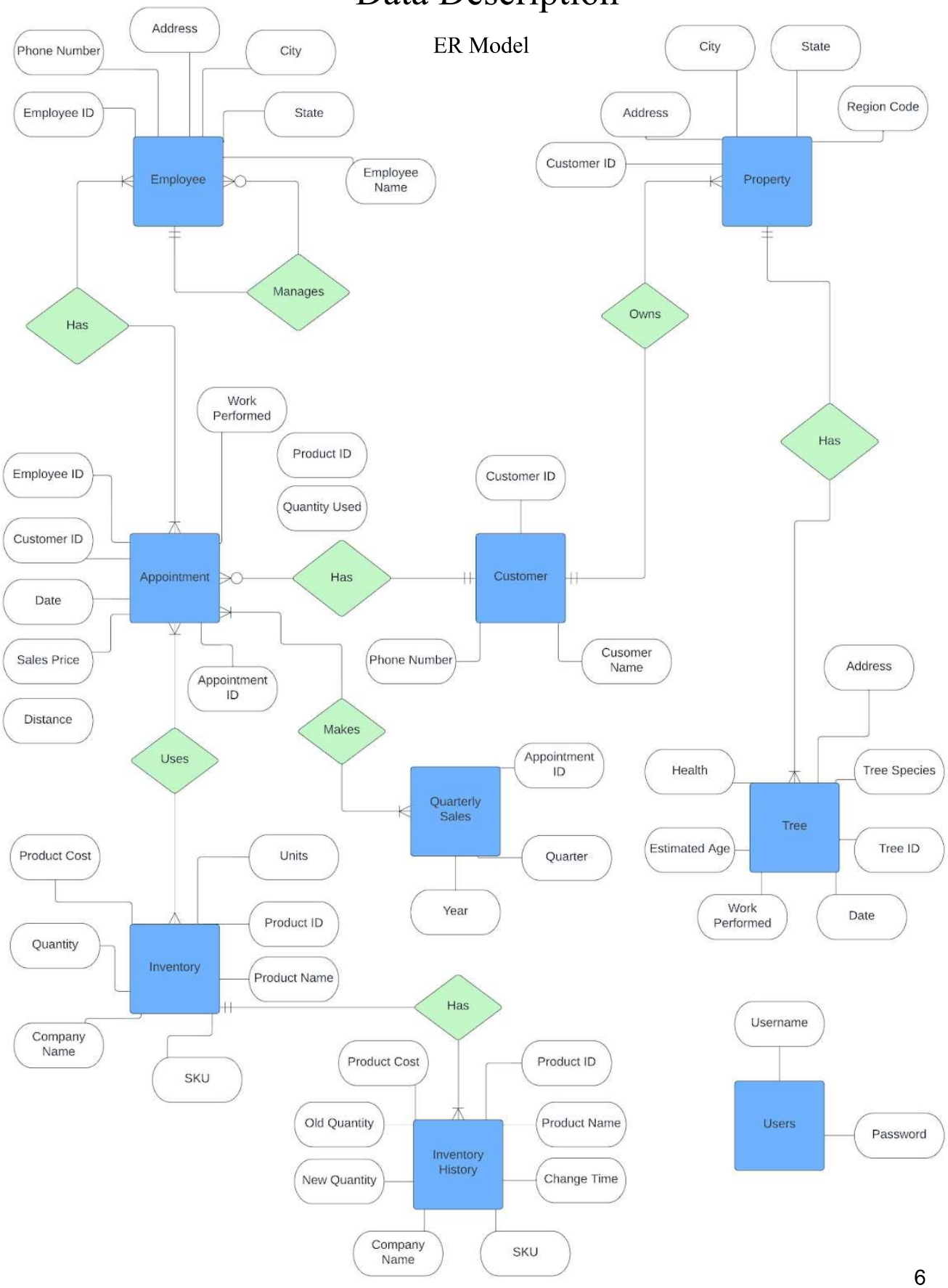
The project proposal, this report, and all code will be submitted in a Google Drive folder. The SQL database and code was created in PGadmin, a postgreSQL client. A connection to the database was made using PGadmin and hosted on AWS. The ER diagram and relational database schema was created using Lucidchart.

The webapp was coded in Python using the Flask package as its web framework. The psycopg2 package was used to connect to the database in the app and create a cursor which performs the CRUD operations (create, read, update, and delete). The frontend of the webapp was coded in HTML using bootstrap as its CSS framework and dynamic data was made available using Jinja2. The webapp was hosted on Heroku and is currently accessible at this link with the username: eastern1 and password: eastern1

A requirements.txt file is included in the Google Drive folder which includes all python packages and versions necessary to run the webapp. The entire project used a combination of Github and Git for version control. Github desktop was used at the initial stages of app development in order to visually observe the progress over time. Once the app was completed, Git was used to clone the github repository and push the required components to Heroku. Heroku logs, the command line, and Git were used to bring the app to its current state of completion.

Data Description

ER Model



Data and Functional Requirements

The database is designed as a record keeping system for this fictional tree company. It is intended to serve as both company records and the backend for a functional webapp. Employees from the company will use the webapp in the field to insert, update, and view some data. Employees will have access to only a small part of the system and the changes they make to the database will be recorded. Managers will interact with the database directly through a PostgreSQL client to run queries and make changes, as well as delete data that was added incorrectly or is no longer relevant. For this section entity types will head paragraphs in **bold** and attributes will be written in *italics.* The database contains a total of 9 tables.

Employee

The employee entity type serves to hold employee information to be used by managers. Most of these entities do not have relationships with other tables but more so serve to hold data about employees for company records. This includes the employee's *phone number, address, city, state* and *name*. The *employee ID* will have a one to many relationship with the entity set **Appointment** . This serves as a way to relate the employee with the work they perform for customers, as well as the sales price of the work.

Appointment

Appointment contains the most relationships of any entity set in the database. It has an *appointment ID* attribute that is unique. It uses *employee ID, customer ID,* and *product ID* as a relationship to **Employee, Customer,** and **Inventory** respectively. The table also contains a

work performed attribute as well as the *date* the work was performed and its *sales price*. Through queries these can be used to calculate **Quarterly Sales** as well as value added by individual employees, high dollar customers, etc. Lastly the attributes *distance* and *quantity used* (of product) are included. *Distance* can be viewed by employees to help plan their route to the appointment. *Quantity used* can be updated by employees in the field through the webapp to help keep track of product inventory in real time.

Customer

This entity set is one of the simplest in the database. Similar to the **Employee** table it only contains information about the customers. This information includes *phone number, customer name,* as well as a *customer ID* to easily keep track of customers and prevent data entry errors where individuals may have similar or the same names.

Property

Each **Customer** is associated with a property in a one to many relationship. A **Property** has an *address, city, state code,* and *region* associated with it. This is used to geographically locate the property as well as help quickly calculate distance from the office.

Tree

Each **Tree** is associated with a **Property** in a many to one relationship using the *address* as a foreign key. The primary key *tree ID* is generated as an identity upon creation of a new entity. This entity set also includes information about individual trees and their attributes as *health, estimated age, work performed, date,* and *tree species*. Work performed and date do not

act as foreign keys to **Appointment** in order to allow workers to insert new entities in the field through the webapp. Afterwards it is the managers duty to clean this data and incorporate it into an entity in the **Appointment** table if necessary. This is useful in case the customers have a tree that needs to be worked on that was not included in the original bid. Or if the workers make an observation on a tree and want to share it with the management when they return to the office.

Inventory

The **Inventory** entity set uses the attribute *product id* as a primary key, it is an auto-incrementing number used only internally for the company. It also uses *company name, SKU,* and *product name* to refer back to each inventory item's identity more globally. This table also includes the attributes *product cost, quantity,* and *units.* These are useful in queries to determine how much inventory is left as well as how much it will cost to replace it. This table is for use in record keeping using a SQL client and not available in the webapp.

Inventory History

Inventory History is a copy of the inventory table for the purpose of recording changes made to product quantities in the field. It contains all of the same attributes but splits *quantity* into *old quantity* and *new quantity*. When an entity in **Inventory** changes it triggers a function that creates an entity in the **Inventory History** table. This contains a copy of the other attributes and records both the old and new quantities of that item. This entity set also includes a *change time* attribute to record when those changes were made.

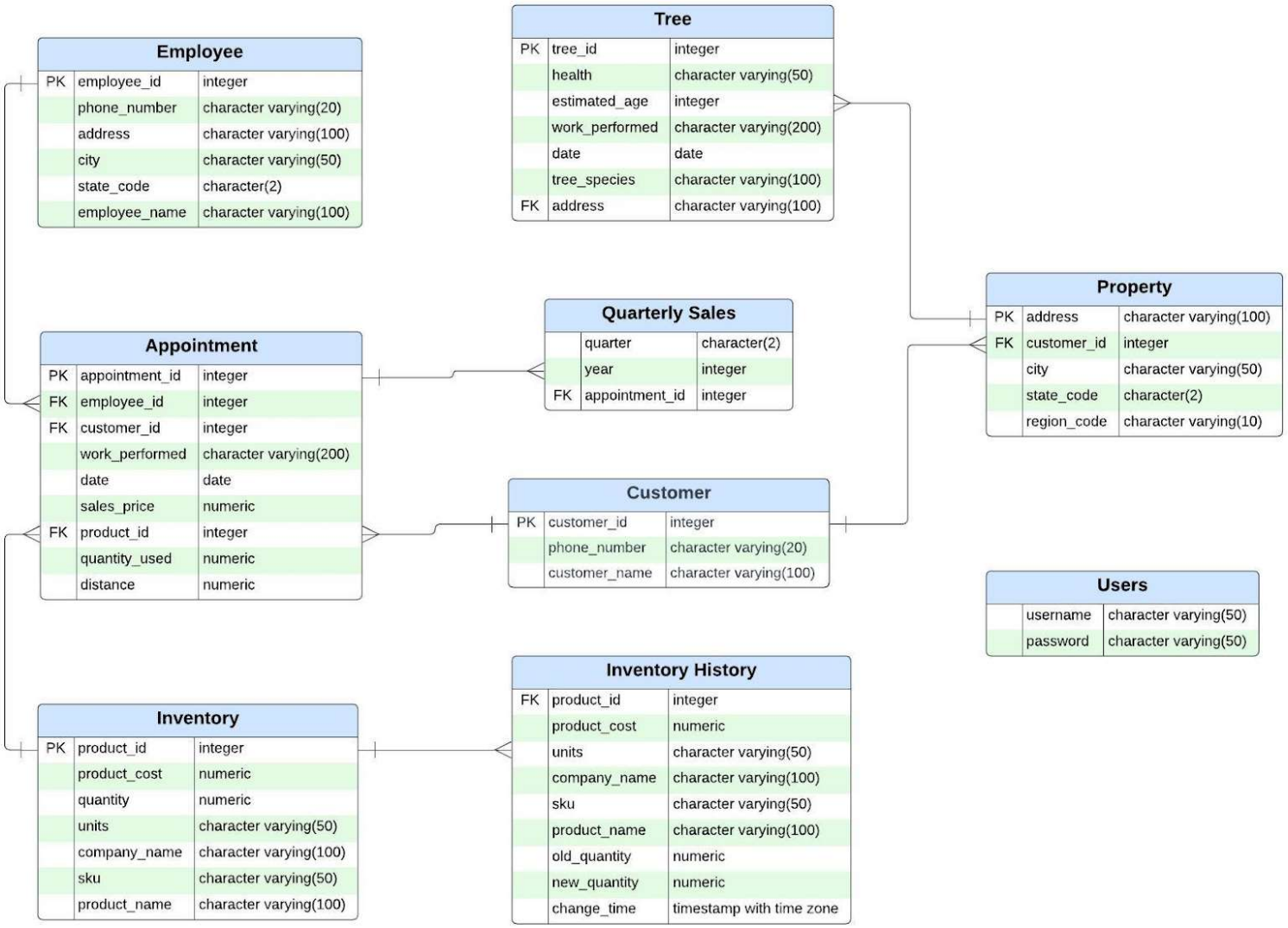
Quarterly Sales

This entity set is also for use in a SQL client. It is mostly for the purpose of running queries to track the company's quarterly sales. It includes the attributes *quarter* and *year*. The *appointment ID* acts as a foreign key to track individual sales made in each quarter.

Users

The final table included in the database is **Users**. This table does not relate to any other table in the schema and is only used to store usernames and passwords. A manager that has access to the database can create a new *username* / *password* combination for each worker that they would like to use the webapp.

Relational Database Schema Diagram



SQL DDL And DML

```
-- Creation of database

CREATE DATABASE gbts;

-- Creation of tables.

CREATE TABLE Employee (
        Employee_ID INT PRIMARY KEY GENERATED ALWAYS AS IDENTITY,
        Phone_number VARCHAR(20),
        Address VARCHAR(100),
        City VARCHAR(50),
        State_code CHAR(2),
        Employee_name VARCHAR(100)
);
CREATE TABLE Customer (
        Customer_ID INT PRIMARY KEY GENERATED ALWAYS AS IDENTITY,
        Phone_number VARCHAR(20),
        Customer_name VARCHAR(100)
);
CREATE TABLE Inventory (
        Product_ID INT PRIMARY KEY GENERATED ALWAYS AS IDENTITY ,
        Product_cost DECIMAL(10, 2),
        Quantity DECIMAL (4,1),
        Units VARCHAR(50),
        Company_name VARCHAR(100),
        SKU VARCHAR(50),
        Product_name VARCHAR(100)
);
CREATE TABLE Appointment (
        Appointment_ID INT PRIMARY KEY GENERATED ALWAYS AS IDENTITY,
        Employee_ID INT,
        Customer_ID INT,
        Work_performed VARCHAR(200),
        Date DATE,
        Sales_price DECIMAL(10, 2),
        Product_ID INT,
        Quantity_used DECIMAL (4,1),
```

```
        Distance DECIMAL (4,1),
        FOREIGN KEY (Employee_ID) REFERENCES Employee(Employee_ID),
        FOREIGN KEY (Customer_ID) REFERENCES Customer(Customer_ID),
        FOREIGN KEY (Product_ID) REFERENCES Inventory (Product_ID)
);
CREATE TABLE Property (
        Address VARCHAR(100) PRIMARY KEY,
        Customer_ID INT,
        City VARCHAR(50),
        State_code CHAR(2),
        Region_code VARCHAR(10),
        FOREIGN KEY (Customer_ID) REFERENCES Customer(Customer_ID)
);
CREATE TABLE Tree (
        Tree_ID INT PRIMARY KEY GENERATED ALWAYS AS IDENTITY,
        Health VARCHAR(50),
        Estimated_age INT,
        Work_performed VARCHAR(200),
        Date DATE,
        Tree_species VARCHAR(100),
        Address VARCHAR(100),
        FOREIGN KEY (Address) REFERENCES Property(Address)
);
CREATE TABLE Quarterly_sales (
        Quarter CHAR(2),
        Year INT,
        Appointment_ID INT,
        FOREIGN KEY (Appointment_ID) REFERENCES Appointment
);
CREATE TABLE Inventory_history (
        Product_ID INT,
        Product_cost DECIMAL(10, 2),
        Units VARCHAR(50),
        Company_name VARCHAR(100),
        SKU VARCHAR(50),
        Product_name VARCHAR(100),
        Old_quantity DECIMAL (4,1),
        New_quantity DECIMAL (4,1),
        Change_time timestamp with time zone,
        FOREIGN KEY (Product_ID) REFERENCES Inventory(Product_ID)
```

```
);
CREATE TABLE users(
        Username VARCHAR(50),
        Password VARCHAR(50)
);


-- Filling the tables with data. Updating and Deleting as necessary

-- Employee

INSERT INTO Employee (Phone_Number, Address, City, State_code, Employee_name)
VALUES      ('1234567890', '123 Main St', 'Reno', 'NV', 'John Doe'),
            ('9876543210', '456 Elm St', 'Reno', 'NV', 'Jane Smith'),
            ('5551234567', '789 Oak St', 'Sparks', 'NV', 'Mike Johnson'),
            ('1112223333', '321 Pine St', 'Sparks', 'NV', 'Sarah Davis'),
            ('4445556666', '654 Maple St', 'Reno', 'NV', 'David Wilson'),
            ('7778889999', '987 Cedar St', 'Carson City', 'NV', 'Emily Thompson'),
            ('2223334444', '741 Birch St', 'Carson City', 'NV', 'Michael Lee'),
            ('9998887777', '852 Spruce St', 'Reno', 'NV', 'Jennifer Brown'),
            ('6665554444', '369 Oak St', 'Reno', 'NV', 'Christopher Taylor'),
            ('3332221111', '963 Elm St', 'Stead', 'NV', 'Jessica Martinez');

UPDATE Employee
SET Address = '345 Maple Ave'
WHERE Employee_name = 'Mike Johnson';

DELETE FROM Employee
WHERE Employee_name = 'Emily Thompson';

-- Customer

INSERT INTO Customer (Phone_number, Customer_name)
VALUES      ('7021111111', 'Bob Johnson'),
            ('7022222222', 'Emily Davis'),
            ('7023333333', 'Michael Smith'),
            ('7024444444', 'Sarah Wilson'),
            ('7025555555', 'David Thompson'),
            ('7026666666', 'Jessica Lee'),
            ('7027777777', 'John Miller'),
```

```
                ('7028888888', 'Amy Taylor'),
                ('7029999999', 'Daniel Anderson'),
                ('7021234567', 'Olivia Martinez');

UPDATE Customer
SET Customer_name = 'Emily Johanssen'
WHERE Customer_ID = 2;

DELETE FROM Customer
WHERE Customer_name = 'Daniel Anderson';

-- Inventory

INSERT INTO Inventory (Product_cost, Quantity, Units, Company_name, SKU, Product_name)
VALUES      (9.99, 10.0, 'Liters', 'TreeCare Products Inc.', 'TC001', 'Rootmax'),
            (9.99, 5.5, 'Liters', 'GreenThumb Tree Care Supplies', 'GT001', 'Tree Fertilizer'),
            (2.99, 100, 'Plugs', 'Evergreen Tree Care Solutions', 'ETC001', 'Arborjet Plugs'),
            (119.99, 15.0, 'Liters', 'ArborPro Tree Services', 'APTS001', 'Bug Destroyer'),
            (49.99, 3.5, 'Liters', 'BranchMaster Tree Care', 'BMTC001', 'Tree Growth
            Hormone'),
            (59.99, 6.0, 'Boxes', 'Nature Tree Care', 'NTC001', 'Tree Insecticide'),
            (39.99, 3, 'Wraps', 'Roots & Shoots Tree Care', 'RSTC001', 'Tree Bark Protector'),
            (19.99, 4.0, 'Bags', 'Canopy Tree Care Products', 'CTCP001', 'Tree Mulch'),
            (9.99, 7, 'Bottles', 'TreeCare Experts', 'TCE001', 'Tree Health Tonic'),
            (99.99, 1.0, 'Kits', 'Forest Guardians Tree Care', 'FGTC001', 'Tree Planting Kit');

UPDATE Inventory
SET Product_cost = 3.99
WHERE Product_name = 'Arborjet Plugs';

DELETE FROM Inventory
WHERE Product_name = 'Tree Planting Kit';

-- Appointment

INSERT INTO Appointment (Employee_ID, Customer_ID, Work_performed, Date, Sales_price,
Product_ID, Quantity_used, Distance)
VALUES      (1, 1, 'Trimming', '2021-01-01', 500, NULL, NULL, 1.5),
            (2, 2, 'Fertilization', '2021-04-01', 200, 1, 1.0, 5),
            (3, 3, 'Injection', '2021-07-01', 500, 4, 2.0, 10),
```

```
                (4, 4, 'Injection', '2021-10-01', NULL, 3, 10, 2),
                (5, 5, 'Fertilization', '2022-01-01', 300, 2, 3.0, 7),
                (7, 6, 'Removal', '2022-04-01', 1000, NULL, NULL, 11),
                (8, 7, 'Injection', '2022-07-01', 600, 4, 1.0, 1),
                (9, 8, 'Injection', '2022-10-01', NULL, 3, 8, 5),
                (10, 10, 'Fertilization', '2023-01-01', 400, 2, 4, 15);

UPDATE Appointment
SET Sales_price = 1200
WHERE Appointment_ID = 6;

DELETE FROM Appointment
WHERE Appointment_ID = 8;

--Property

INSERT INTO Property (Customer_ID, Address, City, State_code, Region_code)
VALUES      (1, '123 Beech St', 'Sparks', 'NV', 'SPARKS'),
            (2, '456 Pine St', 'Reno', 'NV', 'MID'),
            (3, '789 Laurel St', 'Reno', 'NV', 'SW'),
            (4, '987 Elm St', 'Sparks', 'NV', 'SPARKS'),
            (5, '654 Oak St', 'Reno', 'NV', 'MID'),
            (6, '321 Maple St', 'Reno', 'NV', 'SW'),
            (7, '111 Cedar St', 'Sparks', 'NV', 'SPARKS'),
            (8, '222 Walnut St', 'Reno', 'NV', 'MID'),
            (10, '333 Birch St', 'Reno', 'NV', 'SW');

UPDATE Property
SET Address = '333 Birch Ave'
WHERE Address = '333 Birch St';

DELETE FROM property
WHERE Customer_ID = 10;

--Tree

INSERT INTO Tree (Health, Estimated_age, Work_performed, Date, Tree_species, Address)
VALUES      ('Healthy', 20, 'Trimming', '2021-01-01', 'Oak', '123 Beech St'),
            ('Average', 15, 'Fertilization', '2021-04-01', 'Maple', '456 Pine St'),
            ('Poor', 10, 'Injection', '2021-07-01', 'Pine', '789 Laurel St'),
```

```
                ('Healthy', 30, 'Injection', '2021-10-01', 'Spruce', '987 Elm St'),
                ('Average', 25, 'Fertilization', '2022-01-01', 'Birch', '654 Oak St'),
                ('Poor', 10, 'Removal', '2022-04-01', 'Oak', '321 Maple St'),
                ('Healthy', 35, 'Injection', '2022-07-01', 'Maple', '111 Cedar St'),
                ('Average', 20, 'Injection', '2022-10-01', 'Pine', '222 Walnut St'),
                ('Poor', 20, 'Removal', '2023-01-01', 'Maple', '222 Walnut St');

UPDATE Tree
SET Tree_species = 'Oak'
WHERE Tree_ID = 9;

DELETE FROM Tree
WHERE Tree_ID = 9;

-- Quarterly Sales

INSERT INTO Quarterly_sales (Quarter, Year, Appointment_ID)
VALUES      ('Q1', 2021, 1),
            ('Q2', 2021, 2),
            ('Q3', 2021, 3),
            ('Q4', 2021, 4),
            ('Q1', 2022, 5),
            ('Q2', 2022, 6),
            ('Q3', 2022, 7),
            ('Q4', 2022, NULL),
            ('Q1', 2023, 9);

UPDATE Quarterly_sales
SET Appointment_ID = 8
WHERE Appointment_ID = NULL;

DELETE FROM Quarterly_sales
WHERE Quarter = 'Q4' AND Year = 2022;

-- Inventory History. This table is only to be modified by trigger.

INSERT INTO Inventory_history (Product_ID, Product_cost, Units, Company_name, SKU,
Product_name, Old_quantity, New_quantity, Change_time)
VALUES      (1, 9.99, 'Liters', 'TreeCare Products Inc.', 'TC001', 'Rootmax', 8, NULL, NULL),
```

```
        (2, 9.99, 'Liters', 'GreenThumb Tree Care Supplies', 'GT001', 'Tree Fertilizer', 5.5,
        NULL, NULL),
        (3, 2.99, 'Plugs', 'Evergreen Tree Care Solutions', 'ETC001', 'Arborjet Plugs', 100,
        NULL, NULL),
        (4, 119.99, 'Liters', 'ArborPro Tree Services', 'APTS001', 'Bug Destroyer', 15,
        NULL, NULL),
        (5, 49.99, 'Liters', 'BranchMaster Tree Care', 'BMTC001', 'Tree Growth Hormone',
        3.5, NULL, NULL),
        (6, 59.99, 'Boxes', 'Nature Tree Care', 'NTC001', 'Tree Insecticide', 6.0, NULL,
        NULL),
        (7, 39.99, 'Wraps', 'Roots & Shoots Tree Care', 'RSTC001', 'Tree Bark Protector',
        3, NULL, NULL),
        (8, 19.99, 'Bags', 'Canopy Tree Care Products', 'CTCP001', 'Tree Mulch', 4.0,
        NULL, NULL),
        (9, 9.99, 'Bottles', 'TreeCare Experts', 'TCE001', 'Tree Health Tonic', 7, NULL,
        NULL);

CREATE OR REPLACE FUNCTION Record_inventory_changes()
RETURNS TRIGGER AS
$$
BEGIN
    IF NEW.Quantity <> OLD.Quantity THEN
    INSERT INTO Inventory_history (
        Product_ID,
        Product_Cost,
        Units,
        Company_name,
        SKU,
        Product_name,
        Old_quantity,
        New_quantity,
        Change_time
        )
    VALUES
        (
        OLD.Product_ID,
        OLD.Product_Cost,
        OLD.Units,
        OLD.Company_name,
        OLD.SKU,
```

```
                OLD.Product_name,
                OLD.quantity,
                NEW.quantity,
                now()
                );
        END IF;
        RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER inventory_history
AFTER UPDATE
ON Inventory
FOR EACH ROW
EXECUTE PROCEDURE Record_inventory_changes();

-- Users

INSERT INTO Users (Username, Password)
VALUES      ('eastern1', 'eastern1');


-- Create 2 Views on the data

CREATE VIEW Dying_trees AS
                SELECT Tree_ID, Health
                FROM Tree
                WHERE Health = 'Poor';

CREATE VIEW Low_inventory AS
                SELECT Product_ID, Quantity
                FROM Inventory
                WHERE Quantity <= 1;

-- Creation of tree() function.

        -- This allows a user to easily research the attributes of a tree using only its ID.

CREATE OR REPLACE FUNCTION tree(ID INT)
        RETURNS TABLE (
```

```
        tree_health VARCHAR(50),
    tree_age INT,
    tree_work VARCHAR(200),
    work_date DATE,
    species VARCHAR(100)
        )
AS $$
BEGIN
        RETURN QUERY SELECT
                Health,
                Estimated_Age,
                Work_performed,
                Date,
                Tree_species
        FROM
                public.tree
        WHERE
                Tree_ID = ID;
END;
$$
LANGUAGE 'plpgsql';

-- Creation of sales() function.

        -- This allows a manager to easily observe the value created by each individual employee.

CREATE OR REPLACE FUNCTION sales(person INT)
        RETURNS TABLE (
        name VARCHAR(50),
    sales DECIMAL(10,2)
        )
AS $$
BEGIN
        RETURN QUERY SELECT
                Employee_name,
                CAST(AVG(Sales_price) AS DECIMAL(10,2))
        FROM
                public.employee,
                public.appointment
        WHERE
```

```
            employee.Employee_ID = person
            AND appointment.Employee_ID = person
    GROUP BY
            employee.Employee_name;
END;
$$
LANGUAGE 'plpgsql';

-- Creation of Stored Procedure update_inventory()

    -- Updates quantity of inventory based on appointment quantity and product used

CREATE OR REPLACE FUNCTION update_inventory()
RETURNS TRIGGER
AS $$
BEGIN
    IF NEW.Quantity_used <> OLD.Quantity_used THEN
            UPDATE Inventory
            SET Quantity = Quantity - (NEW.Quantity_used - OLD.Quantity_used)
            WHERE Product_ID = NEW.Product_ID;
    END IF;
    RETURN NEW;
END;
$$
LANGUAGE plpgsql;

-- Creation of Trigger

    -- Triggers Stored procedure after appointment quantity has been updated

CREATE OR REPLACE TRIGGER inventory_update
    BEFORE UPDATE OF quantity_used ON appointment
    FOR EACH ROW
    EXECUTE FUNCTION update_inventory();
```

Queries

```
-- See sales made by each employee

SELECT SUM(sales_price), employee_id
FROM appointment
GROUP BY employee_id;

-- Calculate sales price per quarter

SELECT SUM(sales_price), quarter, year
FROM quarterly_sales q
JOIN appointment a ON a.appointment_id = q.appointment_id
GROUP BY year, quarter
ORDER BY year, quarter;

-- Observe what the most popular tree treatment is by quarter

SELECT COUNT(work_performed), work_performed, quarter, year
FROM appointment a
JOIN quarterly_sales q ON q.appointment_id = a.appointment_id
GROUP BY work_performed, year, quarter
ORDER BY year, quarter;

-- Observe what the most common work type is on healthy trees

SELECT COUNT(work_performed), work_performed, health FROM tree
WHERE health = 'Healthy'
GROUP BY work_performed, health;

-- Select the top 3 tree species worked on the most

SELECT tree_species, COUNT(*) AS appointment_count
FROM tree t
JOIN appointment a ON t.date = a.date
GROUP BY tree_species
ORDER BY appointment_count DESC
LIMIT 3;
```

```
-- Select trees that have been removed

SELECT *
FROM appointment
WHERE work_performed ILIKE '%removal%';

-- Calculate the average product cost and quantity for each product and company

SELECT company_name, product_name, AVG(product_cost)::money AS avg_Cost,
ROUND(AVG(quantity),2) AS avg_quantity
FROM inventory
GROUP BY company_name, product_name;

-- Display the top 5 employees who have traveled the most to perform work for customers

SELECT e.employee_id, e.employee_name, SUM(a.distance) AS total_distance
FROM employee e
JOIN appointment a ON e.employee_id = a.employee_id
GROUP BY e.employee_id, e.employee_name
ORDER BY total_distance DESC
LIMIT 5;
```

Analysis

The work was completed following the details outlined in the project proposal. The First week consisted of mapping out the basic relationships needed in the database, creating the ER diagram, and submitting the project proposal. The second week consisted of reviewing the critiques on the proposal and beginning to create the repository on Github. The third week involved completing the SQL components and hosting the database on AWS.

The fourth through sixth weeks were the most labor intensive. A Python file was created in order to program the webapp and styling began. The database was connected to the webapp using the psycopg2 package and the Python file continued to be worked on using the Flask package. Once the webapp was debugged locally it was hosted on Heroku and tested. At the end of the sixth week the final project report began. Week seven consisted of completing the report, the video walkthrough, and moving all files into the shared drive.

Week	Activity Details
1	• Planning and constructing ER diagram. Finishing and submitting proposal for review
2	• Review critiques on proposal and amend as necessary. Start Gihub repo and begin creating Relational Database Schema Diagram and SQL
3	• Complete Diagram and create and fill SQL database, including queries and DDL/DML • Host database on AWS
4-5	• Connect database to Python using packages and code webapp in Flask • Test Webapp and host on Heroku
6-7	• Debug webapp • Create Project walkthrough Video and Final Project Report • Submit Project

Future Work

The biggest hurdle faced in this project was the 7 week timeframe. Another difficulty was the split responsibility of both designing and implementing the database, as well as coding a functioning webapp to perform operations on it, dividing that timeframe even further. Now that both of these are completed, there are some improvements that could be made moving forward.

Firstly, it would make sense to host the database and the app on the same service. Using both AWS and Heroku was an unintended consequence of the steep learning curve of AWS. After technical difficulties slowed production down by two days it was decided to move forward with another hosting service for the webapp. Heroku was found to have a very pleasant user interface and streamlined hosting process. Also Microsoft Azure could be investigated in the future as they offer free services to students.

The other area for improvement would be the webapp itself. With more time it would be wise to create more error handling within the app as well as a more robust user verification process. Session timeout would be another wise feature to add as well as connection pooling to help scale it. Replacing psycopg2 or incorporating the SQLAlchemy package could also be used to streamline some of these processes. Overall the project was completed successfully using the software discussed. Both the webapp and the database have full function and are ready to use by the Great Basin Tree Service.

Templates

Appointment.html

```
{% extends 'base.html' %}

{% block content %}
    <h1>{% block title %} Appointments {% endblock %}</h1>
    <table>
    <div class="col-md-8">
        <table id="example" class="tb" style="width:100%">
            <thead>
                <tr>
                    <td>Appointment ID</td>
                    <td>Employee ID</td>
                    <td>Customer ID</td>
                    <td>Distance</td>
                    <td>Work Performed</td>
                    <td>Date</td>
                    <td>Sales Price</td>
                    <td>Product ID</td>
                    <td>Quantity Used</td>
                </tr>
            </thead>
            <tbody>
                {% for appointment in appointments %}
                <tr>
                    <td>{{appointment.appointment_id}}</td>
                    <td>{{appointment.employee_id}}</td>
                    <td>{{appointment.customer_id}}</td>
                    <td>{{appointment.distance}}</td>
                    <td>{{appointment.work_performed}}</td>
                    <td>{{appointment.date}}</td>
                    <td>{{appointment.sales_price}}</td>
                    <td>{{appointment.product_id}}</td>
                    <td>{{appointment.quantity_used}}
                        {% if appointment.product_id is not none %}
                        <form method="POST" action="/login/appointment">
                            <div>
                                <input type="hidden" name="row_id"
value="{{ appointment.appointment_id }}">
                                <label for="quantity">Quantity
Used</label>
                                <input type="number" min="0" max="10"
step="0.5" name="quantity_used" id="quantity_used" required>
```

```
                                <button class="btn btn-success
float-right" type="submit">Update</button>
                            </div>
                        </form>
                        {% endif %}
                    </td>
                </tr>
                {% endfor %}
            </tbody>
      </table>
    </div>
{% endblock %}
```

Base.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title> {% block title %} {% endblock %} </title>
    <link rel="stylesheet"
href="https://cdn.jsdelivr.net/npm/bootstrap@4.0.0/dist/css/bootstrap.min.
css"
integrity="sha384-Gn5384xqQ1aoWXA+058RXPxPg6fy4IWvTNh0E263XmFcJlSAwiGgFAW/
dAiS6JXm" crossorigin="anonymous">
    <link href='https://fonts.googleapis.com/css?family=Lato:100'
rel='stylesheet' type='text/css'>

    <style>
            .navbar-brand
            {
                font-family: 'Lato', sans-serif;
                color:grey;
                font-size: 60px !important;
                margin: 0px;
            }
              nav a {
                color: #226c38;
                font-size: 2em;
                margin-left: 50px;
                text-decoration: none;
            }

              .dropdown {
                position: relative;
                display: inline-block;
            }
```

```
  .dropdown-content {
    display: none;
    position: absolute;
    background-color: #f9f9f9;
    min-width: 160px;
    box-shadow: 0px 8px 16px 0px rgba(0,0,0,0.2);
    z-index: 1;
}

  .dropdown:hover .dropdown-content {
    display: block;
}

  .dropdown-content a {
    color: black;
    padding: 12px 16px;
    text-decoration: none;
    display: block;
}

  .dropdown-content a:hover {
    background-color: #f1f1f1;
}
  body {
    background-image:
url('https://lh3.googleusercontent.com/qQYmFPu9g_zYz--U-xsOJUtUSmROeyECFQo
lZv8sDvpCYNTiqr5B3DTK3Kj4dMIu87wxrNdOOTLx0RAXHLIDL0wBpPFsv2nJCzI45XFYervbt
HrKB7TQLDSGcXOZouRxe8F6zewj8Vo=w2400');
    background-repeat: no-repeat;
    background-size: cover;
    background-position: center;
}

  .tb {
    border-collapse: collapse; width:300px;
    background-color: rgb(241, 238, 240);
    opacity: 70%;
}
  .tb th, .tb td { padding: 5px; border: solid 1px #777; }
  .tb th { background-color: rgb(230, 173, 215); }

  .form-box {
    border: 1px solid #ccc;
    background-color: rgb(241, 238, 240);
    opacity: 70%;
    padding: 20px;
```

```
    border-radius: 5px;
    width: 300px;
}

  .form-box div {
    margin-bottom: 10px;
}

  .form-box label {
    display: block;
}

  .form-box input[type="text"] {
    width: 100%;
    padding: 5px;
    border: 1px solid #ccc;
    border-radius: 3px;
}

  .form-box input[type="submit"] {
    margin-top: 10px;
    padding: 5px 10px;
    background-color: #4CAF50;
    color: white;
    border: none;
    border-radius: 3px;
    cursor: pointer;
}

  select{
    border:none;
    padding: 10px 20px;
    border-radius:5px;
}

  select:focus{
    outline:none;
}
  .image-box {
    width: 80vw;
    height: 70vh;
    border: 1px solid #000;
    padding: 10px;
    box-sizing: border-box;
}

  .image-box img {
```

```
                max-width: 100%;
                max-height: 100%;
                object-fit: contain;
                display: block;
                margin: 0 auto;
            }
    </style>
</head>
<body>
    <nav class="navbar navbar-expand-lg navbar-light bg-light">
        <a class="navbar-brand" href="#">Great Basin Tree Service</a>
        <button class="navbar-toggler" type="button"
data-toggle="collapse" data-target="#navbarNavDropdown"
aria-controls="navbarNavDropdown" aria-expanded="false" aria-label="Toggle
navigation">
          <span class="navbar-toggler-icon"></span>
        </button>
        <div class="collapse navbar-collapse" id="navbarNavDropdown">
          <ul class="navbar-nav">
            <li class="nav-item active">
              <a class="nav-link" href="{{ url_for('insert') }}">New
Record</a>
            </li>
            <li class="nav-item">
              <a class="nav-link" href="{{ url_for('trees') }}">Trees</a>
            </li>
            <li class="nav-item">
                <a class="nav-link" href="{{ url_for('appointment')
}}">Appointments</a>
            </li>
            <li class="nav-item">
              <a class="nav-link" href="{{ url_for('inventory')
}}">Inventory</a>
            </li>
            <li class="nav-item">
              <a class="nav-link" href="{{ url_for('logout')
}}">Logout</a>
            </li>
          </ul>
        </div>
      </nav>
    <hr>
    <div class="'content">
        {% block content %} {% endblock %}
    </div>
```

```
    <script src="https://code.jquery.com/jquery-3.2.1.slim.min.js"
integrity="sha384-KJ3o2DKtIkvYIK3UENzmM7KCkRr/rE9/Qpg6aAZGJwFDMVNA/GpGFF93
hXpG5KkN" crossorigin="anonymous"></script>
    <script
src="https://cdn.jsdelivr.net/npm/popper.js@1.12.9/dist/umd/popper.min.js"
integrity="sha384-ApNbgh9B+Y1QKtv3Rn7W3mgPxhU9K/ScQsAP7hUibX39j7fakFPskvXu
svfa0b4Q" crossorigin="anonymous"></script>
    <script
src="https://cdn.jsdelivr.net/npm/bootstrap@4.0.0/dist/js/bootstrap.min.js
"
integrity="sha384-JZR6Spejh4U02d8jOt6vLEHfe/JQGiRRSQQxSfFWpi1MquVdAyjUar5+
76PVCmYl" crossorigin="anonymous"></script>
</body>
</html>
```

Index.html

```
{% extends 'base.html' %}

{% block content %}



{% endblock %}
```

Insert_form.html

```
{% extends 'base.html' %}

{% block content %}
    <h1>{% block title %} New Record {% endblock %}</h1>
        <div class="form-box">
            <form method="POST" action="/login/insert">

                <div>
                  <label for="health">Health:</label>
                  <input type="text" name="health"
pattern="^(Healthy|Average|Poor)$" required title="Please enter a valid
value (Healthy, Average, or Poor)">
                </div>

                <div>
                  <label for="estimated_age">Estimated Age:</label>
                  <input type="number" name="estimated_age"
id="estimated_age" required>
                </div>
```

```
                <div>
                  <label for="work_performed">Work Performed:</label>
                  <input type="text" name="work_performed"
id="work_performed" required>
                </div>

                <div>
                  <label for="date">Date:</label>
                  <input type="date" name="date" id="date" required>
                </div>

                <div>
                  <label for="tree_species">Tree Species:</label>
                  <input type="text" name="tree_species" id="tree_species"
maxlength="100" required>
                </div>

                <div class="dropdown">
                    <label for="address">Address:</label>
                    <select name="address" id="address">
                    {% for address in property %}
                    <option value="{{address}}">{{address}}</option>
                    {% endfor %}
                    </select>
                </div>

                <input type="submit" value="Insert">
              </form>
        </div>
        <h1>
{% endblock %}
```

Inventory.html

```
{% extends 'base.html' %}

{% block content %}
    <h1>{% block title %} Inventory {% endblock %}</h1>
    <table>
    <div class="col-md-8">
        <table id="example" class="tb" style="width:100%">
            <thead>
                <tr>
                    <td>Product ID</td>
                    <td>Product Cost</td>
                    <td>Quantity</td>
                    <td>Units</td>
```

```
                    <td>Company Name</td>
                    <td>SKU</td>
                    <td>Product Name</td>
                </tr>
            </thead>
            <tbody>
                {% for inventory in inventory %}
                <tr>
                    <td>{{inventory.product_id}}</td>
                    <td>{{inventory.product_cost}}</td>
                    <td>{{inventory.quantity}}</td>
                    <td>{{inventory.units}}</td>
                    <td>{{inventory.company_name}}</td>
                    <td>{{inventory.sku}}</td>
                    <td>{{inventory.product_name}}</td>
                </tr>
                {% endfor %}
            </tbody>
      </table>
    </div>
{% endblock %}
```

Login.html

```
{% extends 'base.html' %}

{% block content %}

<div class="form-box">
    <h3>Please enter your username and password</h3>
    <form class="form-group" action="{{ url_for('login') }}"
method="post">
        <label>Username</label>
        <input type="text" class="form-control" name="username"
placeholder="Username" id="username" required>
        <label>Password</label>
        <input type="password" class="form-control" name="password"
placeholder="Password" id="password" required>
        <input type="submit" value="Login" class="form-control btn
btn-success " name="">
    </form>
    {% with messages = get_flashed_messages()  %}
    {% if messages %}
    {% for message in messages %}
    <div class="alert alert-success alert-dismissible fade show"
role="alert">
      {{ message }}
```

```
      <button type="button" class="close" data-dismiss="alert"
aria-label="Close">
        <span aria-hidden="true">×</span>
      </button>
    </div>
    {% endfor %}
    {% endif %}
    {% endwith %}
</div>

{% endblock %}
```

Trees.html

```
{% extends 'base.html' %}

{% block content %}
    <h1>{% block title %} Trees {% endblock %}</h1>
    <div class="col-md-8">
        <table id="example" class="tb" style="width:100%">
            <thead>
                <tr>
                    <td>Tree ID</td>
                    <td>Health</td>
                    <td>Estimated Age</td>
                    <td>Work Performed</td>
                    <td>Date</td>
                    <td>Tree Species</td>
                    <td>Address</td>
                </tr>
            </thead>
            <tbody>
                {% for tree in trees %}
                <tr>
                    <td>{{tree.tree_id}}</td>
                    <td>{{tree.health}}</td>
                    <td>{{tree.estimated_age}}</td>
                    <td>{{tree.work_performed}}</td>
                    <td>{{tree.date}}</td>
                    <td>{{tree.tree_species}}</td>
                    <td>{{tree.address}}</td>
                </tr>
                {% endfor %}
            </tbody>
      </table>
    </div>
{% endblock %}
```

Application

```
from flask import Flask, render_template, request, flash, session, flash,
redirect, url_for
import psycopg2
import psycopg2.extras

app = Flask(__name__, template_folder='templates')
app.secret_key = 'bananabread'

def get_db_connection():
    conn = psycopg2.connect(user='masterUsername',
                                password='bananabread',

host='geeby.cpgbioavvyfo.us-west-1.rds.amazonaws.com',
                                port='5432',
                                database='gbts')
    return conn

@app.route('/')
def home():
    if 'loggedin' in session:
        return render_template('insert_form.html',
username=session['username'])
    return redirect(url_for('login'))

@app.route('/login/', methods=['GET', 'POST'])
def login():

    if request.method == 'POST' and 'username' in request.form and
'password' in request.form:

        conn = get_db_connection()
        cur = conn.cursor(cursor_factory=psycopg2.extras.DictCursor)

        username = request.form['username']
        password = request.form['password']

        cur.execute('SELECT * FROM users WHERE username = %s AND password
= %s', (username, password))
        account = cur.fetchone()

        if account:

            session['loggedin'] = True
            session['username'] = account['username']
```

```
            cur.close()
            conn.close()

            return redirect(url_for('insert'))

        else:
            flash('Incorrect username/password')

    return render_template('login.html')

@app.route('/login/logout')
def logout():

    session.pop('loggedin', None)
    session.pop('username', None)

    return redirect(url_for('login'))


@app.route('/login/insert', methods=['GET','POST'])
def insert():

    if 'loggedin' in session:

        if request.method == 'POST':

            conn = get_db_connection()
            cur = conn.cursor(cursor_factory=psycopg2.extras.DictCursor)

            health_value = request.form['health']
            estimated_age_value = request.form['estimated_age']
            work_performed_value = request.form['work_performed']
            date_value = request.form['date']
            tree_species_value = request.form['tree_species']
            address_value = request.form['address']

            cur.execute('INSERT INTO Tree (health, estimated_age,
work_performed, date, tree_species, address) VALUES (%s, %s, %s, %s, %s,
%s)', (health_value, estimated_age_value, work_performed_value,
date_value, tree_species_value, address_value))

            conn.commit()

        def get_addresses():
            conn = get_db_connection()
            cur = conn.cursor(cursor_factory=psycopg2.extras.DictCursor)
            a = "SELECT address FROM Property"
```

```
            cur.execute(a)
            address = [item[0] for item in cur.fetchall()]
            return address

        return render_template('insert_form.html', property =
get_addresses())

    return redirect(url_for('login'))

@app.route('/login/trees')
def trees():

    if 'loggedin' in session:

        conn = get_db_connection()
        cur = conn.cursor(cursor_factory=psycopg2.extras.DictCursor)
        t = "SELECT * FROM Tree"
        cur.execute(t)
        trees = cur.fetchall()
        return render_template('trees.html', trees = trees)

    return redirect(url_for('login'))

@app.route('/login/appointment', methods=['GET', 'POST'])
def appointment():

    if 'loggedin' in session:
        conn = get_db_connection()
        cur = conn.cursor(cursor_factory=psycopg2.extras.DictCursor)
        t = "SELECT * FROM Appointment ORDER BY Appointment_ID"

        if request.method == 'POST':

            row_id = request.form['row_id']
            q = request.form['quantity_used']

            cur.execute("UPDATE Appointment SET Quantity_used = %s WHERE
Appointment_ID = %s", (q, row_id))
            conn.commit()

            flash('Inventory updated successfully')

        cur.execute(t)
        appointments = cur.fetchall()

        return render_template('appointment.html', appointments =
appointments)
```

```
    return redirect(url_for('login'))

@app.route('/login/inventory')
def inventory():

    if 'loggedin' in session:
        conn = get_db_connection()
        cur = conn.cursor(cursor_factory=psycopg2.extras.DictCursor)
        inv = "SELECT * FROM Inventory ORDER BY Product_ID"
        cur.execute(inv)
        inventory = cur.fetchall()
        return render_template('inventory.html', inventory = inventory)

    return redirect(url_for('login'))

if __name__ == "__main__":
    app.run()
```

Application docs

Requirements.txt

```
blinker==1.6.2
botocore==1.29.154
cement==2.8.2
certifi==2023.5.7
charset-normalizer==2.0.12
click==8.1.3
colorama==0.4.3
distlib==0.3.6
docutils==0.16
filelock==3.12.2
Flask==2.3.2
Flask-SQLAlchemy==3.0.3
Flask-WTF==1.1.1
greenlet==2.0.2
gunicorn==20.1.0
idna==3.4
itsdangerous==2.1.2
Jinja2==3.1.2
jmespath==1.0.1
MarkupSafe==2.1.2
pathspec==0.10.1
platformdirs==3.5.3
postgres==4.0
psycopg2==2.9.6
psycopg2-binary==2.9.6
psycopg2-pool==1.1
pyasn1==0.5.0
python-dateutil==2.8.2
PyYAML==5.4.1
requests==2.26.0
rsa==4.7.2
s3transfer==0.6.1
semantic-version==2.8.5
six==1.14.0
SQLAlchemy==2.0.15
termcolor==1.1.0
typing_extensions==4.6.3
urllib3==1.26.16
virtualenv==20.23.0
wcwidth==0.1.9
Werkzeug==2.3.4
```

```
WTForms==3.0.1
```