

DTSC-691

Capstone Database Final Project

2023SU1

By: Matt Dalgetty

Table of Contents

[**Background 3**](#_1xdt50l0e7ua)

[**Problem Objective 4**](#_v3gejleeky88)

[**Software 5**](#_co2ots2e02ax)

[**Data Description 6**](#_8s1x4prk15fo)

[**Data and Functional Requirements 7**](#_y09lxuacy6ol)

[**Relational Database Schema Diagram 11**](#_wdp4fvhyof41)

[**SQL DDL And DML 12**](#_phl3sw6je06g)

[**Queries 22**](#_uug63dc7v47l)

[**Analysis 24**](#_2b8jn5kxmxqn)

[**Future Work 25**](#_tp87ur214oys)

# 

# 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](https://geeby-a122f76d8790.herokuapp.com/login/) 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**

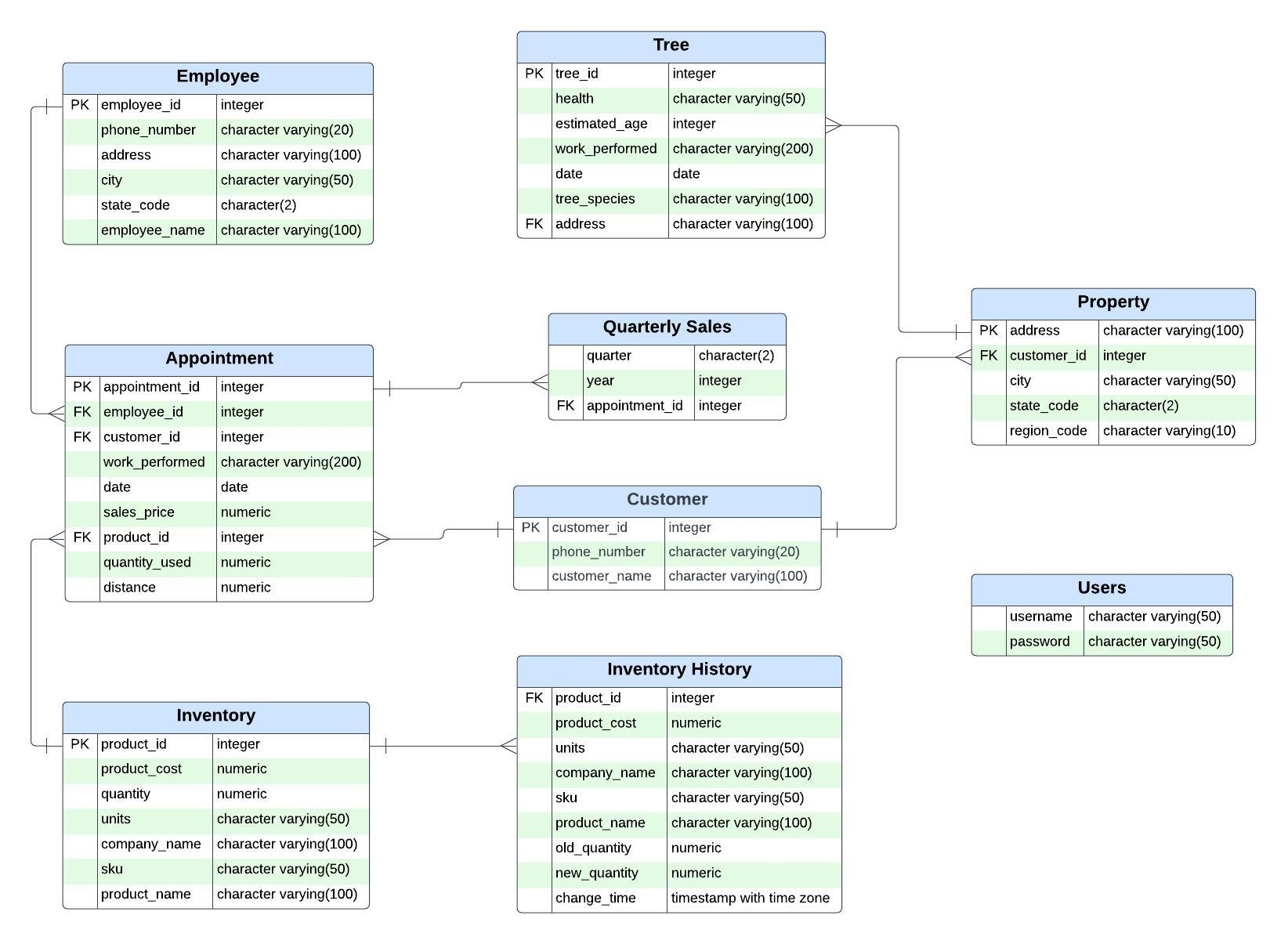
This entity set is also for use in a SQL client. It is mostly for the purpose of running queries to track the company's quarterly sales. It includes the attributes *quarter* and *year*. The *appointment ID* acts as a foreign key to track individual sales made in each quarter.

**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.