***Day 07 Task Allocation: -*** Pan **Shops module**

Creating a basic asset inventory management system for a **Pan Shops module** using **Python** and **PostgreSQL** involves setting up the database, creating tables, and writing Python code to interact with the database. Below is an example code snippet to get you started. This example assumes you have **PostgreSQL** installed and set up with a database ready for use.

**1. Database Setup**

First, you'll need to create the database and tables in **PostgreSQL.**

**Source Code: -**

-- Create a database named 'pan\_shop\_inventory'

CREATE DATABASE pan\_shop\_inventory;

-- Connect to the database

\c pan\_shop\_inventory;

-- Create a table to store asset information

CREATE TABLE assets (

id SERIAL PRIMARY KEY,

name VARCHAR(255) NOT NULL,

category VARCHAR(100) NOT NULL,

purchase\_date DATE NOT NULL,

cost DECIMAL(10, 2) NOT NULL,

status VARCHAR(50) NOT NULL

);

### 2. Python Code for Asset Inventory Management System

Below is the Python code that connects to the PostgreSQL database and allows you to perform basic CRUD (Create, Read, Update, Delete) operations for managing assets.

**Source Code: -**

**import psycopg2**

**from psycopg2 import sql**

**# Database connection settings**

**DB\_HOST = "localhost"**

**DB\_NAME = "pan\_shop\_inventory"**

**DB\_USER = "your\_username" # Replace with your PostgreSQL username**

**DB\_PASSWORD = "your\_password" # Replace with your PostgreSQL password**

**# Connect to the PostgreSQL database**

**conn = psycopg2.connect(**

**host=DB\_HOST,**

**dbname=DB\_NAME,**

**user=DB\_USER,**

**password=DB\_PASSWORD**

**)**

**cur = conn.cursor()**

**# Function to add a new asset**

**def add\_asset(name, category, purchase\_date, cost, status):**

**insert\_query = sql.SQL("""**

**INSERT INTO assets (name, category, purchase\_date, cost, status)**

**VALUES (%s, %s, %s, %s, %s)**

**""")**

**cur.execute(insert\_query, (name, category, purchase\_date, cost, status))**

**conn.commit()**

**print(f"Asset '{name}' added successfully.")**

**# Function to view all assets**

**def view\_assets():**

**cur.execute("SELECT \* FROM assets")**

**rows = cur.fetchall()**

**for row in rows:**

**print(row)**

**# Function to update an asset's status**

**def update\_asset\_status(asset\_id, new\_status):**

**update\_query = sql.SQL("""**

**UPDATE assets**

**SET status = %s**

**WHERE id = %s**

**""")**

**cur.execute(update\_query, (new\_status, asset\_id))**

**conn.commit()**

**print(f"Asset ID '{asset\_id}' updated successfully.")**

**# Function to delete an asset**

**def delete\_asset(asset\_id):**

**delete\_query = sql.SQL("DELETE FROM assets WHERE id = %s")**

**cur.execute(delete\_query, (asset\_id,))**

**conn.commit()**

**print(f"Asset ID '{asset\_id}' deleted successfully.")**

**# Example usage**

**if \_\_name\_\_ == "\_\_main\_\_":**

**# Add a new asset**

**add\_asset('Pan Machine', 'Equipment', '2024-08-26', 50000.00, 'Operational')**

**# View all assets**

**print("Current assets in the inventory:")**

**view\_assets()**

**# Update an asset's status**

**update\_asset\_status(1, 'Under Maintenance')**

**# View updated assets**

**print("Updated assets in the inventory:")**

**view\_assets()**

**# Delete an asset**

**delete\_asset(1)**

**# View assets after deletion**

**print("Assets after deletion:")**

**view\_assets()**

**# Close the database connection**

**cur.close()**

**conn.close()**

### 3. Steps to Run

1. **Install PostgreSQL and psycopg2**:
   * Make sure you have PostgreSQL installed and a database created.
   * Install the psycopg2 library using pip:

**Source Code: -**

pip install psycopg2

1. **Configure Database Credentials**:
   * Replace your\_username and your\_password in the Python script with your PostgreSQL credentials.
2. **Run the Script**:
   * Execute the Python script to interact with the asset inventory management system.

This basic implementation allows you to add, view, update, and delete assets from the inventory. You can extend it by adding more features like search functionality, better error handling, or a user interface.