**Day 09 Task Allocation**

Creating a comprehensive Asset Inventory Management System involves several key components: connecting to a PostgreSQL database, creating tables, performing CRUD (Create, Read, Update, Delete) operations, and generating reports. Here's a simplified version of the system using Python and PostgreSQL:

**Step 1: Install Required Libraries**

First, make sure you have the necessary libraries installed:

**Bash Source Code**

pip install psycopg2

### Step 2: Connect to PostgreSQL Database

Create a connection to your PostgreSQL database.

Python Source Code:-

import psycopg2

from psycopg2 import sql

def connect\_db():

try:

connection = psycopg2.connect(

dbname="your\_database",

user="your\_user",

password="your\_password",

host="localhost",

port="5432"

)

return connection

except Exception as e:

print(f"Error connecting to the database: {e}")

return None

### Step 3: Create Tables

Create tables for storing asset information, transactions, and reports.

**Python Source Code:-**

def create\_tables(connection):

with connection.cursor() as cursor:

cursor.execute("""

CREATE TABLE IF NOT EXISTS assets (

asset\_id SERIAL PRIMARY KEY,

asset\_name VARCHAR(255) NOT NULL,

asset\_type VARCHAR(255),

quantity INT NOT NULL,

purchase\_date DATE,

cost NUMERIC(10, 2),

description TEXT

);

""")

cursor.execute("""

CREATE TABLE IF NOT EXISTS transactions (

transaction\_id SERIAL PRIMARY KEY,

asset\_id INT REFERENCES assets(asset\_id),

transaction\_date DATE NOT NULL,

transaction\_type VARCHAR(255) NOT NULL,

quantity INT NOT NULL

);

""")

connection.commit()

print("Tables created successfully")

connection = connect\_db()

if connection:

create\_tables(connection)

connection.close()

### Step 4: Add Assets

Function to add new assets to the inventory.

**Python Source Code:-**

def add\_asset(connection, asset\_name, asset\_type, quantity, purchase\_date, cost, description):

with connection.cursor() as cursor:

cursor.execute("""

INSERT INTO assets (asset\_name, asset\_type, quantity, purchase\_date, cost, description)

VALUES (%s, %s, %s, %s, %s, %s);

""", (asset\_name, asset\_type, quantity, purchase\_date, cost, description))

connection.commit()

print(f"Asset '{asset\_name}' added successfully.")

connection = connect\_db()

if connection:

add\_asset(connection, 'Pan Counter', 'Equipment', 5, '2024-08-01', 1500.00, 'Counter for Pan Shop')

connection.close()

### Step 5: Record Transactions

Function to record asset transactions.

**Python Source Code:-**

def record\_transaction(connection, asset\_id, transaction\_date, transaction\_type, quantity):

with connection.cursor() as cursor:

cursor.execute("""

INSERT INTO transactions (asset\_id, transaction\_date, transaction\_type, quantity)

VALUES (%s, %s, %s, %s);

""", (asset\_id, transaction\_date, transaction\_type, quantity))

cursor.execute("""

UPDATE assets SET quantity = quantity + %s WHERE asset\_id = %s;

""", (quantity if transaction\_type == 'add' else -quantity, asset\_id))

connection.commit()

print("Transaction recorded successfully.")

connection = connect\_db()

if connection:

record\_transaction(connection, 1, '2024-08-10', 'add', 3) # Example: Adding 3 more counters

connection.close()

### Step 6: Generate Reports

Functions to generate daily, weekly, monthly, and yearly reports.

**Python Source Code :-**

import datetime

def generate\_report(connection, period='daily'):

with connection.cursor() as cursor:

if period == 'daily':

cursor.execute("""

SELECT \* FROM assets WHERE purchase\_date = CURRENT\_DATE;

""")

elif period == 'weekly':

cursor.execute("""

SELECT \* FROM assets WHERE purchase\_date >= CURRENT\_DATE - INTERVAL '7 days';

""")

elif period == 'monthly':

cursor.execute("""

SELECT \* FROM assets WHERE purchase\_date >= CURRENT\_DATE - INTERVAL '1 month';

""")

elif period == 'yearly':

cursor.execute("""

SELECT \* FROM assets WHERE purchase\_date >= CURRENT\_DATE - INTERVAL '1 year';

""")

assets = cursor.fetchall()

for asset in assets:

print(asset)

connection = connect\_db()

if connection:

print("Daily Report:")

generate\_report(connection, 'daily')

print("Weekly Report:")

generate\_report(connection, 'weekly')

print("Monthly Report:")

generate\_report(connection, 'monthly')

print("Yearly Report:")

generate\_report(connection, 'yearly')

connection.close()

### Step 7: Main Function

Finally, you can create a main function to integrate all the above functionalities.

**Python Source Code:-**

**def main():**

**connection = connect\_db()**

**if connection:**

**create\_tables(connection)**

**add\_asset(connection, 'Pan Counter', 'Equipment', 5, '2024-08-01', 1500.00, 'Counter for Pan Shop')**

**record\_transaction(connection, 1, '2024-08-10', 'add', 3)**

**print("Daily Report:")**

**generate\_report(connection, 'daily')**

**print("Weekly Report:")**

**generate\_report(connection, 'weekly')**

**print("Monthly Report:")**

**generate\_report(connection, 'monthly')**

**print("Yearly Report:")**

**generate\_report(connection, 'yearly')**

**connection.close()**

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

**main()**

### Notes:

* This code assumes you have a PostgreSQL database set up and running.
* Modify dbname, user, password, host, and port in connect\_db() with your database credentials.
* The asset\_id in record\_transaction should match the asset\_id in the assets table.
* Add error handling and validation as needed.

This script is a basic implementation and can be expanded with additional features like user authentication, more detailed reporting, and a GUI interface if needed.