CASE STUDY:
ASSET MANAGEMENT

Table Of Contents

ABSTRACT

The Digital Asset Management System is a comprehensive application designed to streamline the management, tracking, and maintenance of organizational assets. Built using Python with a strong emphasis on object-oriented principles, the system integrates with a MySQL database to store and retrieve asset-related data efficiently. This project includes key functionalities like asset allocation, maintenance tracking, reservation handling, and exception management and unit testing. The project follows a structured approach, incorporating SQL interactions, control flow statements, loops, collections, and exception handling, while ensuring code reusability and modularity through unit testing.

INTRODUCTION

In modern organizations, managing physical assets such as laptops, vehicles, and equipment is crucial for operational efficiency. The Digital Asset Management System provides a structured solution to track asset allocation, maintenance, and reservations while minimizing manual errors. This project leverages Python's object-oriented programming (OOP), database connectivity (MySQL), and exception handling to create a robust and scalable system. The application follows a menu-driven approach, allowing users to perform operations such as adding, updating, allocating, and maintaining assets seamlessly.

The system allows administrators to:

- Add, update, and delete asset records.
- Allocate and deallocate assets to employees.
- Schedule and log maintenance activities.
- Reserve assets for future use.

PURPOSE OF THE PROJECT

The primary objective of this project was to develop a centralized system that eliminates the inefficiencies of manual asset tracking. Traditional methods often lead to misplaced equipment, maintenance oversights, and allocation conflicts. This system provides real-time visibility into asset status, ensuring proper utilization and timely maintenance. Beyond basic tracking, the application incorporates exception handling to prevent invalid operations and includes comprehensive unit tests to verify functionality. The result is a reliable tool that reduces administrative overhead while improving asset accountability across the organization. The objectives are:

1. **Efficient Asset Management:** Maintain a database for tracking assets, their allocation, and maintenance history.
2. **Automated Tracking:** Reduce manual intervention by automating asset allocation, deallocation, and reservation processes.
3. **Exception Handling:** Implement custom exceptions (AssetNotFoundException, AssetNotMaintainedException) to handle invalid operations.
4. **Database Integration:** Use **MySQL** to store and retrieve asset-related data securely.
5. **Unit Testing:** Ensure system reliability by validating functionalities through **PyTest** test cases

SCOPE OF THE PROJECT

The Digital Asset Management System covers the complete lifecycle of organizational assets, from acquisition to retirement. Core functionalities include adding new assets to the system, updating their details, and removing obsolete items. The allocation module tracks which employee is responsible for each asset, while the maintenance component logs service history and associated costs. A reservation system prevents scheduling conflicts by allowing employees to book assets in advance. While currently designed for single-organization use, the system's modular architecture allows for future expansion to support additional features like barcode scanning or multi-location tracking.

SOFTWARE USED

The development of this system utilized PyCharm as the primary integrated development environment, providing powerful tools for writing, debugging, and testing Python code. MySQL served as the backend database, offering reliable data storage and retrieval capabilities. Key Python libraries included mysql-connector-python for database connectivity and unittest for creating and running test cases. This combination of tools ensured an efficient development workflow while maintaining code quality.

SYSTEM ARCHITECTURE

The application follows a layered architecture that separates concerns and promotes maintainability. At the foundation are entity classes that model real-world assets, employees, and maintenance records without containing business logic. The data access layer handles all database operations, implementing the core functionality defined in service interfaces. Utility classes manage database connections and configuration, while custom exceptions handle error scenarios gracefully. The presentation layer provides a user-friendly menu interface that guides administrators through available operations. This clear separation of responsibilities makes the system both flexible and easy to extend.

TASK 1: SQL SCHEMA

ERD:

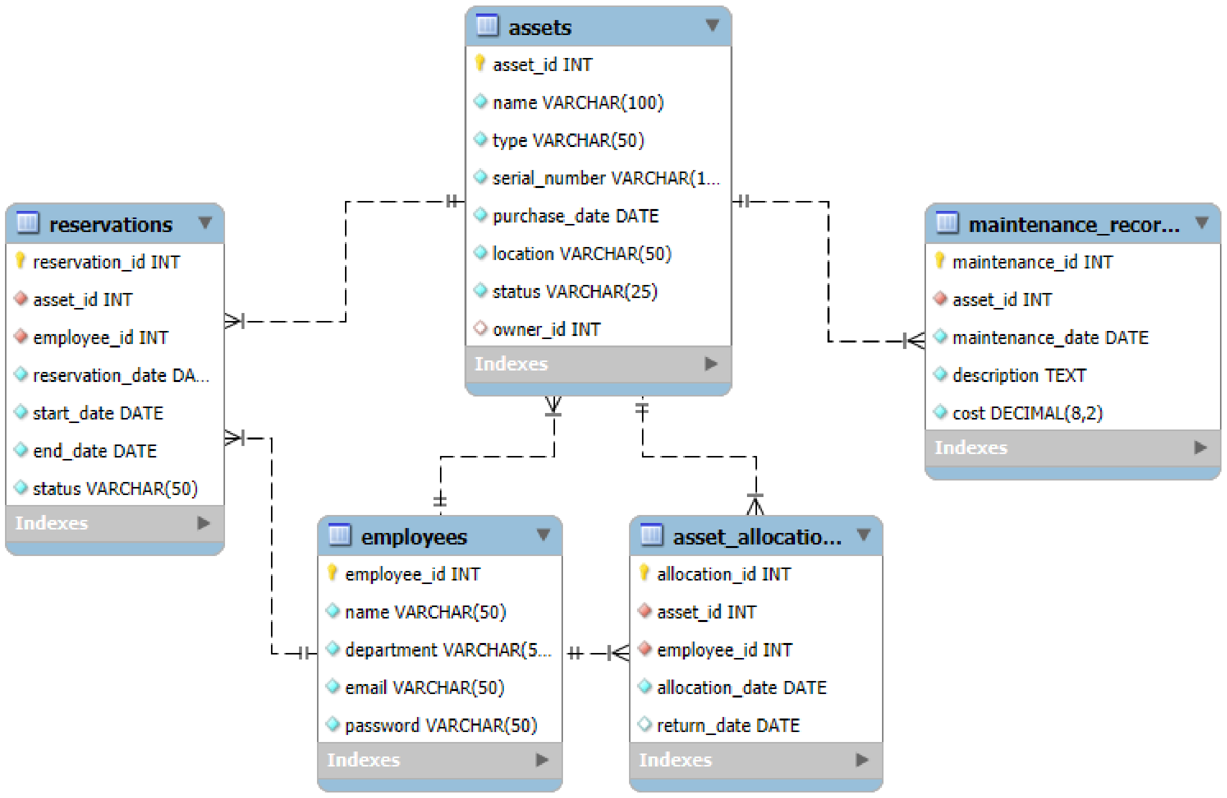


TABLE CREATION:

EMPLOYEE TABLE:

```
create table employees(
employee_id int primary key auto_increment,
name varchar(50) not null,
department varchar(50) not null,
email varchar(50) unique not null,
password varchar(50) not null);

insert into employees (name, department, email, password) values
('Tony Roger', 'IT', 'tonystark2019@gmail.com', 'password12'),
('Son Heung-min', 'HR', 'tottenham@gmail.com', 'trophyless'),
('Emma Ruth', 'HR', 'emmaruth@gmail.com', 'geniuss123'),
('Marie Yeager', 'Research', 'marieyeager@gmail.com', 'rumbling123'),
('Alicia Sabari', 'Marketing', 'alicia2k3@gmail.com', 'ramenos456'),
('Chris Ron', 'Finance', 'cristiano@gmail.com', 'cristia935'),
('Eren Curie', 'IT', 'erencurie@example.com', 'radium123'),
('Fathima kakarot', 'Marketing', 'fathima271@gmail.com', 'kamehameha'),
('John Parker', 'Research', 'albert2003@gmail.com', 'sinister12'),
('Rahul Bachchan', 'Operations', 'rahulbachchan@gmail.com', 'rockbottom');
```

ASSET TABLE:

```
create table assets (
asset_id int primary key auto_increment,
name varchar(100) not null,
type varchar(50) not null,
serial_number varchar(10) unique not null,
purchase_date date not null,
location varchar(50) not null,
status varchar(25) not null,
owner_id int,
foreign key (owner_id) references employees(employee_id) on delete set null);
select * from assets;

insert into assets (name, type, serial_number, purchase_date, location, status, owner_id) values
('Laptop', 'Electronics', 'sn123456', '2025-01-02', 'office', 'in use', 1),
('Projector', 'Equipment', 'sn789012', '2024-11-15', 'meeting room', 'available', null),
('Company Car', 'Vehicle', 'sn345678', '2019-11-10', 'garage', 'in use', 2),
('Printer', 'Electronics', 'sn901234', '2024-09-20', 'office', 'under maintenance', 3),
('Drone', 'Equipment', 'sn567890', '2024-12-05', 'warehouse', 'in use', 4),
('Raspberry Pi Kit', 'electronics', 'sn123789', '2024-03-01', 'data center', 'in use', 5),
('Camera', 'Electronics', 'sn456123', '2025-02-12', 'studio', 'under maintenance', 5),
('CPU', 'Electronics', 'sn789456', '2024-11-30', 'office', 'in use', 6),
('Security System', 'Electronics', 'sn321654', '2024-12-18', 'office', 'in use', 7),
('Macbook Pro', 'Electronics', 'sn987321', '2025-01-01', 'office', 'in use', 8);
```

MAINTENANCE RECORD:

```
create table maintenance_records (
maintenance_id int primary key auto_increment,
asset_id int not null,
maintenance_date date not null,
description text not null,
cost decimal(8, 2) not null,
foreign key (asset_id) references assets(asset_id) on delete cascade);

insert into maintenance_records (asset_id, maintenance_date, description, cost) values
(1, '2025-03-15', 'replaced keyboard and battery', 550.00),
(3, '2025-02-20', 'routine oil change and tire rotation', 300.00),
(4, '2025-01-25', 'replaced toner cartridge and fixed paper jam', 180.00),
(5, '2025-04-10', 'calibrated sensors and replaced propellers', 800.00),
(6, '2025-03-01', 'updated firmware and tested functionality', 50.00),
(7, '2025-02-28', 'cleaned lens and replaced damaged cables', 150.00),
(8, '2025-03-05', 'replaced thermal paste and upgraded cooling system', 200.00),
(9, '2025-04-01', 'updated security software and replaced faulty sensors', 150.00),
(10, '2025-03-20', 'replaced screen and upgraded RAM', 425.00);
```

ASSET ALLOCATION:

```
create table asset_allocations(
allocation_id int primary key auto_increment,
asset_id int not null,
employee_id int not null,
allocation_date date not null,
return_date date,
foreign key (asset_id) references assets(asset_id) on delete cascade,
foreign key (employee_id) references employees(employee_id) on delete cascade);

insert into asset_allocations (asset_id, employee_id, allocation_date, return_date) values
(1, 1, '2025-01-05', null),
(2, 2, '2024-11-20', '2024-12-20'),
(3, 3, '2019-11-15', null),
(4, 4, '2024-09-25', '2024-10-25'),
(5, 5, '2024-12-10', null),
(6, 6, '2024-03-05', '2024-04-05'),
(7, 7, '2025-02-15', null),
(8, 8, '2024-12-01', null),
(9, 9, '2024-12-20', null),
(10, 10, '2025-01-05', null);
```

RESERVATIONS

```
create table reservations(
reservation_id int primary key auto_increment,
asset_id int not null,
employee_id int not null,
reservation_date date not null,
start_date date not null,
end_date date not null,
status varchar(50) not null,
foreign key (asset_id) references assets(asset_id) on delete cascade,
foreign key (employee_id) references employees(employee_id) on delete cascade);

insert into reservations (asset_id, employee_id, reservation_date, start_date, end_date, status) values
(1, 2, '2025-01-01', '2025-01-10', '2025-01-15', 'approved'),
(2, 3, '2024-11-10', '2024-11-20', '2024-11-25', 'approved'),
(3, 4, '2024-09-15', '2024-09-25', '2024-09-30', 'cancelled'),
(4, 5, '2024-08-01', '2024-08-10', '2024-08-15', 'approved'),
(5, 6, '2024-12-01', '2024-12-10', '2024-12-15', 'pending'),
(6, 7, '2024-02-25', '2024-03-05', '2024-03-10', 'approved'),
(7, 8, '2025-01-20', '2025-01-25', '2025-01-30', 'cancelled'),
(8, 9, '2024-11-25', '2024-12-01', '2024-12-05', 'approved'),
(9, 10, '2024-12-10', '2024-12-15', '2024-12-20', 'pending'),
(10, 1, '2024-12-20', '2024-12-25', '2024-12-30', 'approved');
```

Task 2: ENTITY PACKAGE

1) EMPLOYEE

```
class Employee:
    def __init__(self, employee_id=None, name=None, department=None, email=None,
password=None):
        self.__employee_id = employee_id
        self.__name = name
        self.__department = department
        self.__email = email
        self.__password = password

    @property
    def employee_id(self):
        return self.__employee_id

    @property
    def name(self):
        return self.__name

    @property
    def department(self):
        return self.__department

    @property
    def email(self):
        return self.__email

    @property
    def password(self):
        return self.__password

    @employee_id.setter
    def employee_id(self, value):
        self.__employee_id = value

    @name.setter
    def name(self, value):
        self.__name = value

    @department.setter
    def department(self, value):
        self.__department = value

    @email.setter
    def email(self, value):
        self.__email = value

    @password.setter
    def password(self, value):
        self.__password = value
```

2) ASSET:

```
class Asset:
    def __init__(self, asset_id=None, name=None, asset_type=None, serial_number=None,
                 purchase_date=None, location=None, status=None, owner_id=None):
        self.__asset_id = asset_id
        self.__name = name
        self.__asset_type = asset_type
        self.__serial_number = serial_number
        self.__purchase_date = purchase_date
        self.__location = location
        self.__status = status
        self.__owner_id = owner_id

    @property
    def asset_id(self):
        return self.__asset_id
    @property
    def name(self):
        return self.__name

    @property
    def asset_type(self):
        return self.__asset_type

    @property
    def serial_number(self):
        return self.__serial_number

    @property
    def purchase_date(self):
        return self.__purchase_date

    @property
    def location(self):
        return self.__location

    @property
    def status(self):
        return self.__status

    @property
    def owner_id(self):
        return self.__owner_id

    @asset_id.setter
    def asset_id(self, value):
        self.__asset_id = value

    @name.setter
    def name(self, value):
        self.__name = value

    @asset_type.setter
    def asset_type(self, value):
        self.__asset_type = value
    @serial_number.setter
    def serial_number(self, value):
        self.__serial_number = value

    @purchase_date.setter
    def purchase_date(self, value):
        self.__purchase_date = value

    @location.setter
    def location(self, value):
        self.__location = value

    @status.setter
    def status(self, value):
        self.__status = value

    @owner_id.setter
    def owner_id(self, value):
        self.__owner_id = value
```

3) ASSET ALLOCATION

```
class AssetAllocation:
    def __init__(self, allocation_id=None, asset_id=None, employee_id=None,
                 allocation_date=None, return_date=None):
        self.__allocation_id = allocation_id
        self.__asset_id = asset_id
        self.__employee_id = employee_id
        self.__allocation_date = allocation_date
        self.__return_date = return_date

    @property
    def allocation_id(self):
        return self.__allocation_id

    @property
    def asset_id(self):
        return self.__asset_id
```

```
@property
def employee_id(self):
    return self.__employee_id

@property
def allocation_date(self):
    return self.__allocation_date

@property
def return_date(self):
    return self.__return_date

@allocation_id.setter
def allocation_id(self, value):
    self.__allocation_id = value

@asset_id.setter
def asset_id(self, value):
    self.__asset_id = value

@employee_id.setter
def employee_id(self, value):
    self.__employee_id = value

@allocation_date.setter
def allocation_date(self, value):
    self.__allocation_date = value

@return_date.setter
def return_date(self, value):
    self.__return_date = value
```

4) MAINTENANCE RECORDS

```
class MaintenanceRecord:
    def __init__(self, maintenance_id=None, asset_id=None, maintenance_date=None,
                 description=None, cost=None):
        self.__maintenance_id = maintenance_id
        self.__asset_id = asset_id
        self.__maintenance_date = maintenance_date
        self.__description = description
        self.__cost = cost

    @property
    def maintenance_id(self):
        return self.__maintenance_id

    @property
    def asset_id(self):
        return self.__asset_id

    @property
    def maintenance_date(self):
        return self.__maintenance_date

    @property
    def description(self):
        return self.__description

    @property
    def cost(self):
        return self.__cost

    @maintenance_id.setter
    def maintenance_id(self, value):
        self.__maintenance_id = value

    @asset_id.setter
    def asset_id(self, value):
        self.__asset_id = value

    @maintenance_date.setter
    def maintenance_date(self, value):
        self.__maintenance_date = value

    @description.setter
    def description(self, value):
        self.__description = value

    @cost.setter
    def cost(self, value):
        self.__cost = value
```

5) RESERVATIONS

```
class Reservation:
    def __init__(self, reservation_id=None, asset_id=None, employee_id=None,
                 reservation_date=None, start_date=None, end_date=None, status=None):
        self.__reservation_id = reservation_id
        self.__asset_id = asset_id
        self.__employee_id = employee_id
        self.__reservation_date = reservation_date
        self.__start_date = start_date
        self.__end_date = end_date
        self.__status = status

    @property
    def reservation_id(self):
        return self.__reservation_id

    @property
    def asset_id(self):
        return self.__asset_id

    @property
    def employee_id(self):
        return self.__employee_id

    @property
    def reservation_date(self):
        return self.__reservation_date

    @property
    def start_date(self):
        return self.__start_date

    @property
    def end_date(self):
        return self.__end_date
```

```
@property
def status(self):
    return self.__status

@reservation_id.setter
def reservation_id(self, value):
    self.__reservation_id = value

@asset_id.setter
def asset_id(self, value):
    self.__asset_id = value

@employee_id.setter
def employee_id(self, value):
    self.__employee_id = value

@reservation_date.setter
def reservation_date(self, value):
    self.__reservation_date = value

@start_date.setter
def start_date(self, value):
    self.__start_date = value

@end_date.setter
def end_date(self, value):
    self.__end_date = value

@status.setter
def status(self, value):
    self.__status = value
```

TASK 3: DAO

1) ABSTRACT CLASS

```
from abc import ABC, abstractmethod
class AssetManagementService(ABC):

   @abstractmethod
   def add_asset(self, name, asset_type, serial_number, purchase_date, location, status,
owner_id):
       pass

   @abstractmethod
   def update_asset(self, asset_id, name=None, location=None, status=None):
       pass

   @abstractmethod
   def delete_asset(self, asset_id):
       pass

   @abstractmethod
   def allocate_asset(self, asset_id, employee_id, allocation_date,return_date):
       pass

   @abstractmethod
   def deallocate_asset(self, asset_id):
       pass

   @abstractmethod
   def perform_maintenance(self, asset_id, maintenance_date, description, cost):
       pass

   @abstractmethod
   def reserve_asset(self, asset_id, employee_id, reservation_date, start_date, end_date, status):
       pass

   @abstractmethod
   def withdraw_reservation(self, reservation_id):
       pass
```

MAIN DAO:

```
import mysql.connector
from util.DBConnection import DBConnection
from myexception.exceptions import AssetNotFoundException
from myexception.exceptions import AssetNotMaintainException
from datetime import date, timedelta

class AssetManagementServiceImpl:
    def __init__(self):
        self.db = DBConnection()

    def add_asset(self, name, asset_type, serial_number, purchase_date, location, status,
owner_id):
        conn = None
        cursor = None
        try:
            conn = self.db.get_connection()
            cursor = conn.cursor()
            query = ("insert into assets (name, type, serial_number, purchase_date, location, status, owner_id) values
(%s, %s, %s, %s, %s, %s, %s)")
            cursor.execute(query, (name, asset_type, serial_number,
                                   purchase_date, location, status, owner_id))
            conn.commit()
            asset_id = cursor.lastrowid
            print(f"asset added successfully! asset id: {asset_id}")
            return cursor.lastrowid
        except mysql.connector.Error as e:
            print(f"database error: {e}")
            return None
        finally:
            if cursor is not None:
                cursor.close()
            if conn is not None:
                conn.close()

    def update_asset(self, asset_id, new_location, new_status):
        conn = None
        cursor = None
        try:
            conn = self.db.get_connection()
            cursor = conn.cursor()
            query = "update assets set location = %s, status = %s where asset_id = %s"
            cursor.execute(query, (new_location, new_status, asset_id))
            if cursor.rowcount == 0:
                raise AssetNotFoundException("asset id not found.")
            conn.commit()
            print("asset updated successfully!")
            return True
        except mysql.connector.Error as e:
            print(f"database error: {e}")
            return False
        finally:
            if cursor is not None:
                cursor.close()
            if conn is not None:
                conn.close()
```

def delete_asset(self, asset_id):

```
    conn = None
    cursor = None
    try:
        conn = self.db.get_connection()
        cursor = conn.cursor()
        query = "delete from assets where asset_id = %s"
        cursor.execute(query, (asset_id,))
        if cursor.rowcount == 0:
            raise AssetNotFoundException("asset id not found.")
        conn.commit()
        print("asset deleted successfully!")
        return True
    except mysql.connector.Error as e:
        print(f"database error: {e}")
        return False
    finally:
        if cursor is not None:
            cursor.close()
        if conn is not None:
            conn.close()
```

def allocate_asset(self, asset_id, employee_id, allocation_date):

```
    conn = None
    cursor = None
    try:
        conn = self.db.get_connection()
        cursor = conn.cursor(dictionary=True)
        cursor.execute("""
            select status, purchase_date
            from assets
            where asset_id = %s
        """, (asset_id,))
        asset = cursor.fetchone()

        if not asset:
            raise AssetNotFoundException(f"asset id {asset_id} not found.")
        if asset['status'].lower() != 'available':
            print(f"cannot allocate asset - current status: {asset['status']}")
            return False

        purchase_date = asset['purchase_date']
        two_years_ago = date.today() - timedelta(days=365 * 2)

        if purchase_date < two_years_ago:
            cursor.execute("""
                select 1 from maintenance_records
                where asset_id = %s
                and maintenance_date >= %s
                limit 1
            """, (asset_id, two_years_ago))

            if not cursor.fetchone():
                raise AssetNotMaintainException(
                    f"asset {asset_id} hasn't been maintained in the last 2 years")

        cursor.execute("""
            update assets
            set status = 'allocated', owner_id = %s
            where asset_id = %s
        """, (employee_id, asset_id))

        cursor.execute("""
            insert into asset_allocations
            (asset_id, employee_id, allocation_date)
            values (%s, %s, %s)
        """, (asset_id, employee_id, allocation_date))

        conn.commit()
        print(f"asset {asset_id} allocated successfully to employee {employee_id}!")
        return True

    except mysql.connector.Error as e:
        print(f"database error: {e}")
        if conn:
            conn.rollback()
        return False
    finally:
        if cursor:
            cursor.close()
        if conn:
            conn.close()
```

def deallocate_asset(self, asset_id):

```
    conn = None
```

```
cursor = None
try:
    conn = self.db.get_connection()
    cursor = conn.cursor()
    check_query = "select asset_id from assets where asset_id = %s"
    cursor.execute(check_query, (asset_id,))
    if not cursor.fetchone():
        raise AssetNotFoundException("asset id not found.")
    query = "update assets set status = 'available', owner_id = null where asset_id = %s"
    cursor.execute(query, (asset_id,))
    conn.commit()
    print("asset deallocated successfully!")
    return True
except mysql.connector.Error as e:
    print(f"database error: {e}")
    return False
finally:
    if cursor is not None:
        cursor.close()
    if conn is not None:
        conn.close()
```

def perform_maintenance(self, asset_id, maintenance_date, description, cost):

```
conn = None
cursor = None
try:
    conn = self.db.get_connection()
    cursor = conn.cursor()

    cursor.execute("""
        select status from assets
        where asset_id = %s
    """, (asset_id,))
    row = cursor.fetchone()

    if not row:
        raise AssetNotFoundException("asset id not found.")

    status = row[0].lower()

    if status == "under maintenance":
        raise AssetNotMaintainException("asset is already under maintenance.")

    if status == "reserved":
        raise AssetNotMaintainException("cannot perform maintenance on reserved asset.")

    cursor.execute("""
        update assets
```

```
            set status = 'under maintenance'
            where asset_id = %s
        """, (asset_id,))

        cursor.execute("""
            insert into maintenance_records
            (asset_id, maintenance_date, description, cost)
            values (%s, %s, %s, %s)
        """, (asset_id, maintenance_date, description, cost))

        conn.commit()
        print("maintenance recorded successfully!")
        return True

    except mysql.connector.Error as e:
        print(f"database error: {e}")
        if conn:
            conn.rollback()
        return False
    finally:
        if cursor:
            cursor.close()
        if conn:
            conn.close()
```

def reserve_asset(self, asset_id, employee_id, reservation_date, start_date, end_date):

```
    conn = None
    cursor = None
    try:
        conn = self.db.get_connection()
        cursor = conn.cursor()
        check_query = "select status from assets where asset_id = %s"
        cursor.execute(check_query, (asset_id,))
        row = cursor.fetchone()
        if not row:
            raise AssetNotFoundException("asset id not found.")
        if row[0].lower() != 'available':
            print(f"cannot reserve asset - current status: {row[0]}")
            return False
        update_query = "update assets set status = 'reserved' where asset_id = %s"
        cursor.execute(update_query, (asset_id,))
        reservation_query = """insert into reservations
                    (asset_id, employee_id, reservation_date, start_date, end_date, status)
                    values (%s, %s, %s, %s, %s, 'reserved')"""
        cursor.execute(reservation_query, (asset_id, employee_id, reservation_date, start_date, end_date))
        conn.commit()
        reservation_id = cursor.lastrowid
        print(f"asset reserved successfully! reservation id: {reservation_id}")
        return True
    except mysql.connector.Error as e:
        print(f"database error: {e}")
        return False
    finally:
        if cursor is not None:
            cursor.close()
        if conn is not None:
            conn.close()
```

def withdraw_reservation(self, reservation_id):

```
    conn = None
    cursor = None
    try:
        conn = self.db.get_connection()
        cursor = conn.cursor()
        get_query = """select r.asset_id, a.status
                    from reservations r
                    join assets a on r.asset_id = a.asset_id
                    where r.reservation_id = %s"""
        cursor.execute(get_query, (reservation_id,))
        reservation = cursor.fetchone()
        if not reservation:
            raise AssetNotFoundException("reservation id not found.")
        asset_id, current_status = reservation
        if current_status.lower() != 'reserved':
            print(f"cannot withdraw reservation - asset is not reserved (current status: {current_status})")
            return False
        update_asset_query = "update assets set status = 'available' where asset_id = %s"
        cursor.execute(update_asset_query, (asset_id,))
        update_reservation_query = "update reservations set status = 'withdrawn' where reservation_id = %s"
        cursor.execute(update_reservation_query, (reservation_id,))
        conn.commit()
        print("reservation withdrawn successfully! asset is now available.")
        return True
    except mysql.connector.Error as e:
        print(f"database error: {e}")
        return False
    finally:
        if cursor is not None:
            cursor.close()
        if conn is not None:
            conn.close()
```

TASK 4: DATABASE CONNECTION

DBCONNECTION.PY

```
import mysql.connector
from util.DBPropertyUtil import get_db_properties

class DBConnection:
    @staticmethod
    def get_connection():
        db_props = get_db_properties()
        return mysql.connector.connect(
            host=db_props["host"],
            port=db_props["port"],
            user=db_props["user"],
            password=db_props["password"],
            database=db_props["database"]
        )
```

DBPropertyUtil

```
import configparser
import os

def get_db_properties():
    config = configparser.ConfigParser()
    properties_file = os.path.join(os.path.dirname(__file__), "db.properties")

    if not os.path.exists(properties_file):
        raise FileNotFoundError(f"The DB {properties_file} file isn't there")

    config.read(properties_file)

    try:
        return {
            "host": config.get("DEFAULT", "db.host"),
            "port": config.get("DEFAULT", "db.port"),
            "user": config.get("DEFAULT", "db.user"),
            "password": config.get("DEFAULT", "db.password"),
            "database": config.get("DEFAULT", "db.name"),}
    except Exception as e:
        raise Exception(f"Error reading database properties: {e}")
```

db.properties

```
[DEFAULT]
db.host=localhost
db.port=3306
db.user=root
db.password=root
db.name=assetmanagement
```

TASK 5: EXCEPTIONS

```
class AssetNotFoundException(Exception):
    def __init__(self, message="Asset not found"):
        self.message = message
        super().__init__(self.message)

class AssetNotMaintainException(Exception):
    def __init__(self, message="Asset is not properly maintained"):
        self.message = message
        super().__init__(self.message)
```

TASK 6: MAIN

```
from dao.asset_management_service_impl import AssetManagementServiceImpl
from myexception.exceptions import AssetNotFoundException
from myexception.exceptions import AssetNotMaintainException

class AssetManagementApp:
    def __init__(self):
        self.service = AssetManagementServiceImpl()

    def display_menu(self):
        print("\n========== Asset Management System ================")
        print("1. Add Asset")
        print("2. Update Asset")
        print("3. Delete Asset")
        print("4. Allocate Asset")
        print("5. Deallocate Asset")
        print("6. Perform Maintenance")
        print("7. Reserve Asset")
        print("8. Withdraw Reservation")
        print("9. Exit")

    def run(self):
        while True:
            self.display_menu()
            choice = input("Enter your choice: ")
        try:
            if choice == "1":
                name = input("Enter asset name: ")
                asset_type = input("Enter asset type: ")
                serial_number = input("Enter serial number: ")
                purchase_date = input("Enter purchase date (YYYY-MM-DD): ")
                location = input("Enter location: ")
                status = input("Enter status (in use / decommissioned / under maintenance): ")
                owner_id = input("Enter owner ID: ")
                self.service.add_asset(name, asset_type, serial_number, purchase_date, location,
status, owner_id)

            elif choice == "2":
                asset_id = int(input("Enter asset ID to update: "))
                new_location = input("Enter new location: ")
                new_status = input("Enter new status(available/ in use / decommissioned / under
maintenance): ")
                self.service.update_asset(asset_id, new_location, new_status)

            elif choice == "3":
                asset_id = int(input("Enter asset ID to delete: "))
                self.service.delete_asset(asset_id)

            elif choice == "4":
                asset_id = int(input("Enter asset ID to allocate: "))
                employee_id = int(input("Enter employee ID: "))
                allocation_date = input("Enter allocation date (YYYY-MM-DD): ")
                self.service.allocate_asset(asset_id, employee_id, allocation_date)

            elif choice == "5":
                asset_id = int(input("Enter asset ID to deallocate: "))
                self.service.deallocate_asset(asset_id)

            elif choice == "6":
                asset_id = int(input("Enter asset ID for maintenance: "))
                maintenance_date = input("Enter maintenance date (YYYY-MM-DD): ")
                description = input("Enter maintenance description: ")
                cost = float(input("Enter maintenance cost: "))
                self.service.perform_maintenance(asset_id, maintenance_date, description, cost)
            elif choice == "7":
                asset_id = int(input("Enter asset ID to reserve: "))
                employee_id = int(input("Enter employee ID: "))
                reservation_date = input("Enter reservation date (YYYY-MM-DD): ")
                start_date = input("Enter start date (YYYY-MM-DD): ")
                end_date = input("Enter end date (YYYY-MM-DD): ")
                self.service.reserve_asset(asset_id, employee_id, reservation_date, start_date,
end_date)

            elif choice == "8":
                reservation_id = int(input("Enter reservation ID to withdraw: "))
                self.service.withdraw_reservation(reservation_id)

            elif choice == "9":
                print("Exiting the application...")
                break

            else:
                print("Invalid choice! Please try again.")

        except AssetNotFoundException as e:
            print(f"Error: {e}")

        except AssetNotMaintainException as e:
            print(f"Error: {e}")

        except Exception as e:
            print(f"An unexpected error occurred: {e}")

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

TASK 7: TESTING

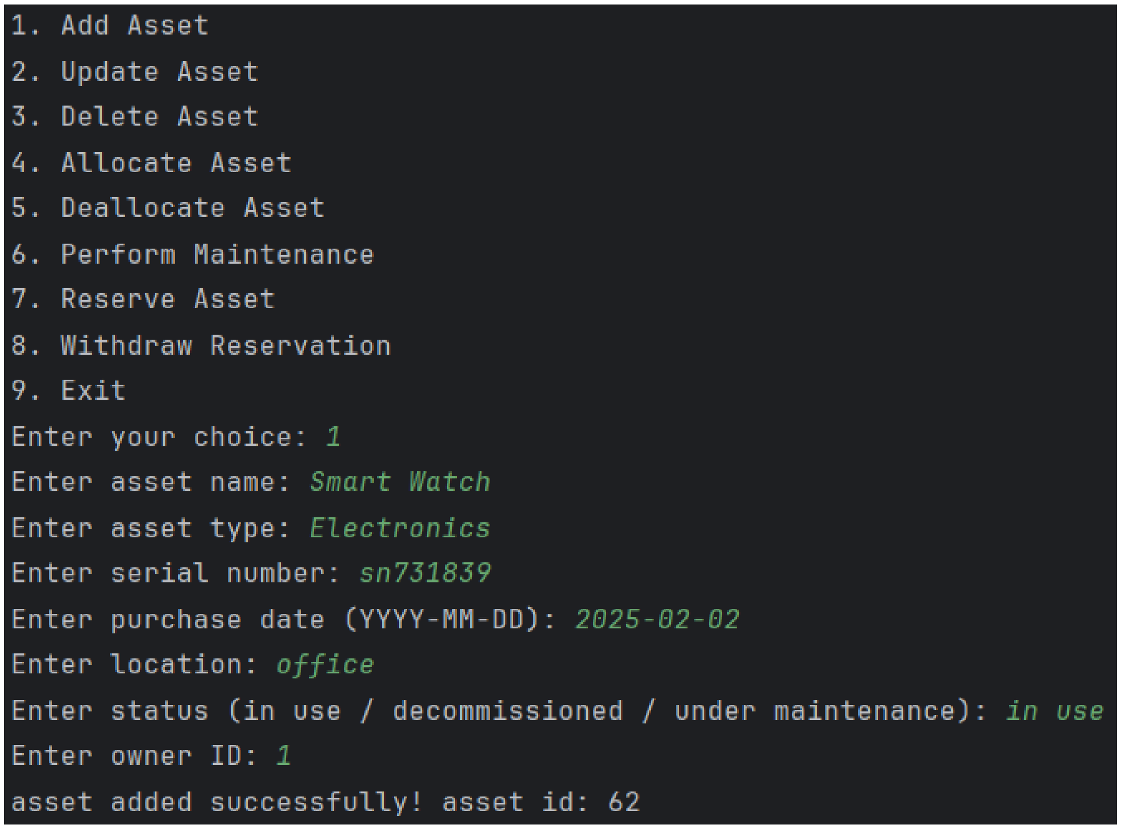
```
import unittest
from dao.asset_management_service_impl import AssetManagementServiceImpl
from myexception.exceptions import AssetNotFoundException, AssetNotMaintainException
import mysql.connector
from datetime import date, timedelta
```

```
class TestAssetManagementSystem(unittest.TestCase):
    @classmethod
    def setUpClass(cls):
        cls.service = AssetManagementServiceImpl()
        cls.test_asset_id = None
        cls.test_employee_id = 1
        cls.test_reservation_id = None

    def test_1_add_asset_success(self):
        name = "Test Asset"
        asset_type = "Test Equipment"
        serial_number = "TEST12"
        purchase_date = date.today().strftime('%Y-%m-%d')
        location = "Test Location"
        status = "available"
        owner_id = None

        asset_id = self.service.add_asset(
            name, asset_type, serial_number,
            purchase_date, location, status, owner_id
        )

        self.assertIsNotNone(asset_id)
        TestAssetManagementSystem.test_asset_id = asset_id

        conn = None
        cursor = None
        try:
            conn = self.service.db.get_connection()
            cursor = conn.cursor()
            cursor.execute("select * from assets where asset_id = %s", (asset_id,))
            result = cursor.fetchone()
            self.assertIsNotNone(result)
            self.assertEqual(result[1], name)
            self.assertEqual(result[6].lower(), "available")
        finally:
            if cursor: cursor.close()
            if conn: conn.close()

    def test_2_reserve_asset_success(self):
        if not hasattr(TestAssetManagementSystem, 'test_asset_id'):
            self.skipTest("No asset available for reservation test")

        employee_id = self.test_employee_id
        reservation_date = date.today().strftime('%Y-%m-%d')
        start_date = (date.today() + timedelta(days=1)).strftime('%Y-%m-%d')
        end_date = (date.today() + timedelta(days=7)).strftime('%Y-%m-%d')

        result = self.service.reserve_asset(
            TestAssetManagementSystem.test_asset_id,
            employee_id,
            reservation_date,
            start_date,
            end_date
        )

        self.assertTrue(result)

        conn = None
        cursor = None
        try:
            conn = self.service.db.get_connection()
            cursor = conn.cursor()
            cursor.execute("select status from assets where asset_id = %s",
(TestAssetManagementSystem.test_asset_id,))
            asset_status = cursor.fetchone()[0]
            self.assertEqual(asset_status.lower(), "reserved")
        finally:
            if cursor: cursor.close()
            if conn: conn.close()

    def test_3_perform_maintenance_on_reserved_asset(self):
        asset_id = self.service.add_asset(
            "Test Asset", "Equipment", "TEST1",
            "2023-01-01", "Location", "available", None
        )
        self.assertIsNotNone(asset_id)

        conn = None
        cursor = None
        try:
            conn = self.service.db.get_connection()
            cursor = conn.cursor()
            cursor.execute("select 1 from assets where asset_id = %s", (asset_id,))
            self.assertTrue(cursor.fetchone())
        finally:
            if cursor: cursor.close()
            if conn: conn.close()

        reserve_result = self.service.reserve_asset(
            asset_id,
            self.test_employee_id,
            date.today().strftime('%Y-%m-%d'),
            (date.today() + timedelta(days=1)).strftime('%Y-%m-%d'),
            (date.today() + timedelta(days=7)).strftime('%Y-%m-%d')
        )
        self.assertTrue(reserve_result)

        with self.assertRaises(AssetNotMaintainException):
            self.service.perform_maintenance(
                asset_id,
                date.today().strftime('%Y-%m-%d'),
                "Test maintenance",
                100.00
            )

        self.service.delete_asset(asset_id)

    def test_4_asset_not_found_exception(self):
        invalid_asset_id = 9999

        with self.assertRaises(AssetNotFoundException):
            self.service.update_asset(invalid_asset_id, "New Location", "available")

        with self.assertRaises(AssetNotFoundException):
            self.service.delete_asset(invalid_asset_id)

        with self.assertRaises(AssetNotFoundException):
            self.service.perform_maintenance(invalid_asset_id, "2023-01-01", "Test", 100.00)
```

```
def test_5_asset_not_maintained_exception(self):
    old_date = (date.today() - timedelta(days=365 * 3)).strftime('%Y-%m-%d')
    asset_id = self.service.add_asset(
        "Old Unmaintained Asset",
        "Equipment",
        "OLD13",
        old_date,
        "Storage",
        "available",
        None
    )

    self.assertIsNotNone(asset_id)

    with self.assertRaises(AssetNotMaintainException):
        self.service.allocate_asset(
            asset_id,
            self.test_employee_id,
            date.today().strftime('%Y-%m-%d')
        )

    self.service.delete_asset(asset_id)

@classmethod
def tearDownClass(cls):
    if hasattr(cls, 'test_asset_id') and cls.test_asset_id:
        conn = None
        cursor = None
        try:
            conn = cls.service.db.get_connection()
            cursor = conn.cursor()
            cursor.execute("delete from reservations where asset_id = %s", (cls.test_asset_id,))
            cursor.execute("delete from assets where asset_id = %s", (cls.test_asset_id,))
            conn.commit()
        except Exception as e:
            if conn: conn.rollback()
        finally:
            if cursor: cursor.close()
            if conn: conn.close()
if __name__ == '__main__':
    unittest.main()
```

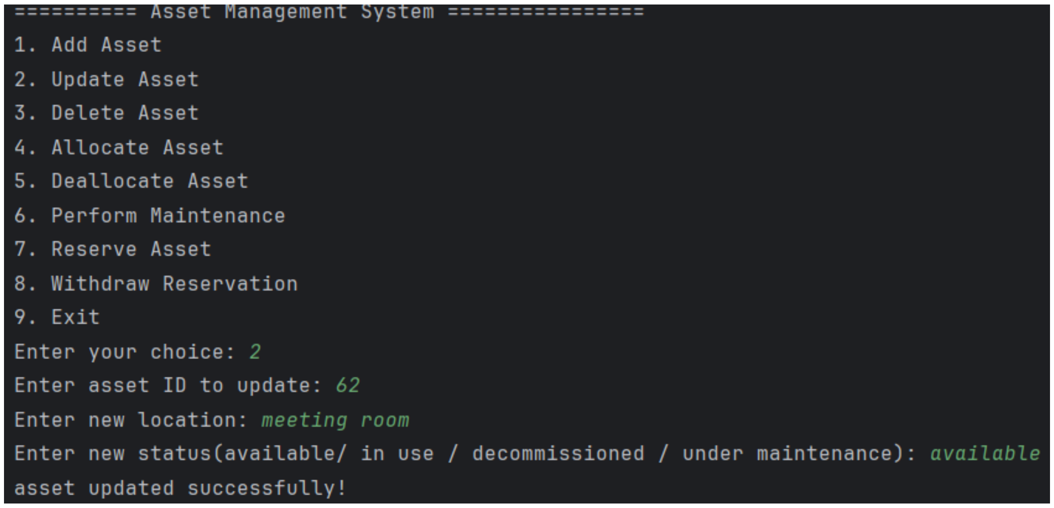
IMPLEMENTATION AND WORKING:

1) ADD ASSET:



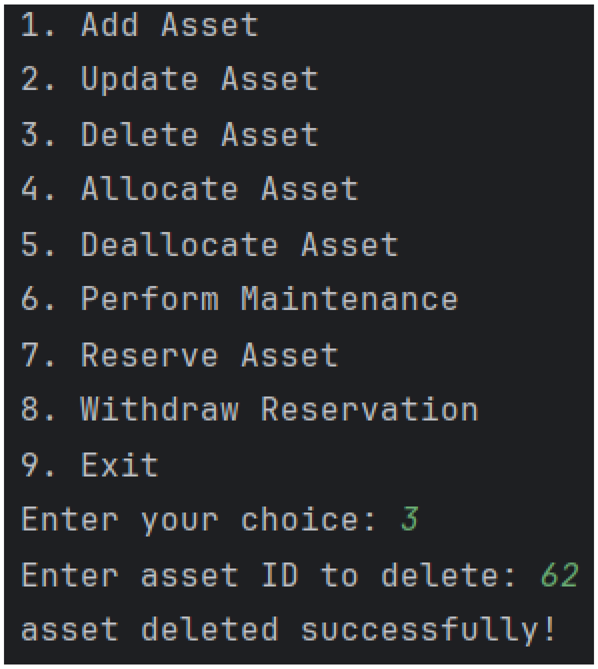
```
1. Add Asset
2. Update Asset
3. Delete Asset
4. Allocate Asset
5. Deallocate Asset
6. Perform Maintenance
7. Reserve Asset
8. Withdraw Reservation
9. Exit
Enter your choice: 1
Enter asset name: Smart Watch
Enter asset type: Electronics
Enter serial number: sn731839
Enter purchase date (YYYY-MM-DD): 2025-02-02
Enter location: office
Enter status (in use / decommissioned / under maintenance): in use
Enter owner ID: 1
asset added successfully! asset id: 62
```

2) UPDATE ASSET



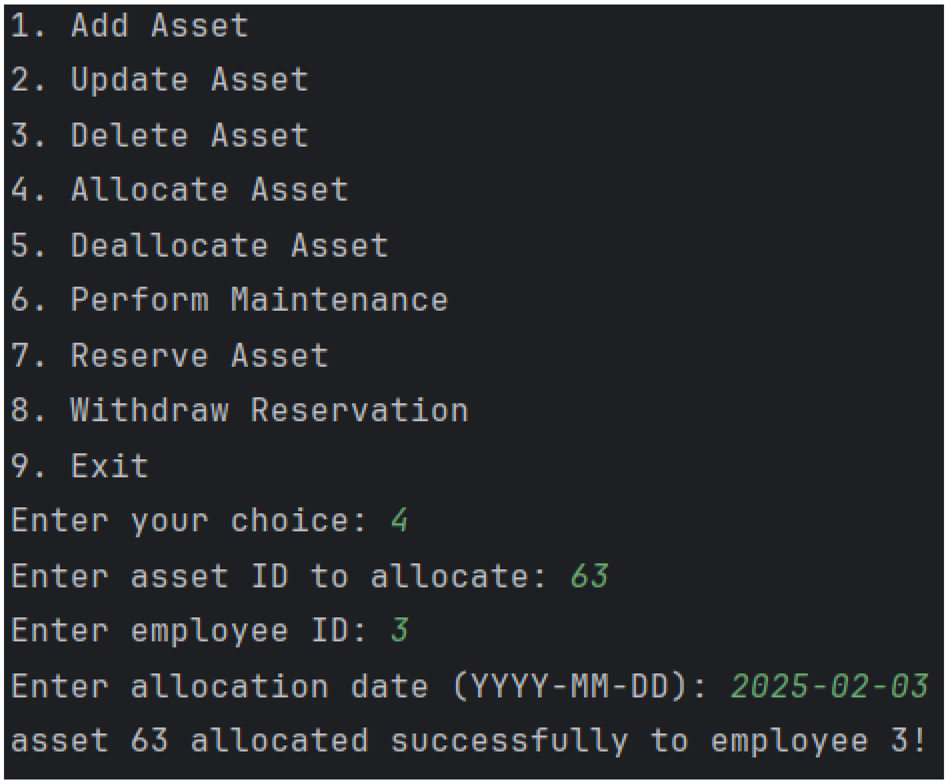
```
========== Asset Management System ================
1. Add Asset
2. Update Asset
3. Delete Asset
4. Allocate Asset
5. Deallocate Asset
6. Perform Maintenance
7. Reserve Asset
8. Withdraw Reservation
9. Exit
Enter your choice: 2
Enter asset ID to update: 62
Enter new location: meeting room
Enter new status(available/ in use / decommissioned / under maintenance): available
asset updated successfully!
```

3) **DELETE ASSET**



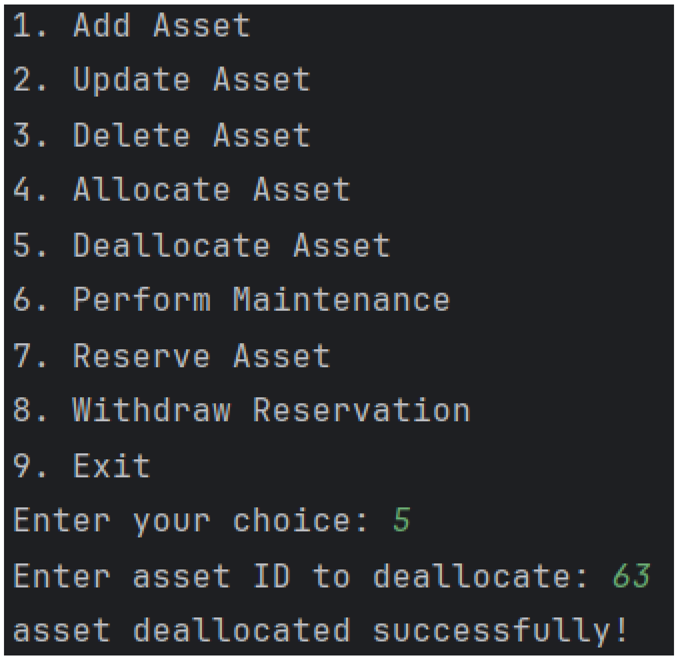
```
1. Add Asset
2. Update Asset
3. Delete Asset
4. Allocate Asset
5. Deallocate Asset
6. Perform Maintenance
7. Reserve Asset
8. Withdraw Reservation
9. Exit
Enter your choice: 3
Enter asset ID to delete: 62
asset deleted successfully!
```

4) **ALLOCATE ASSET**



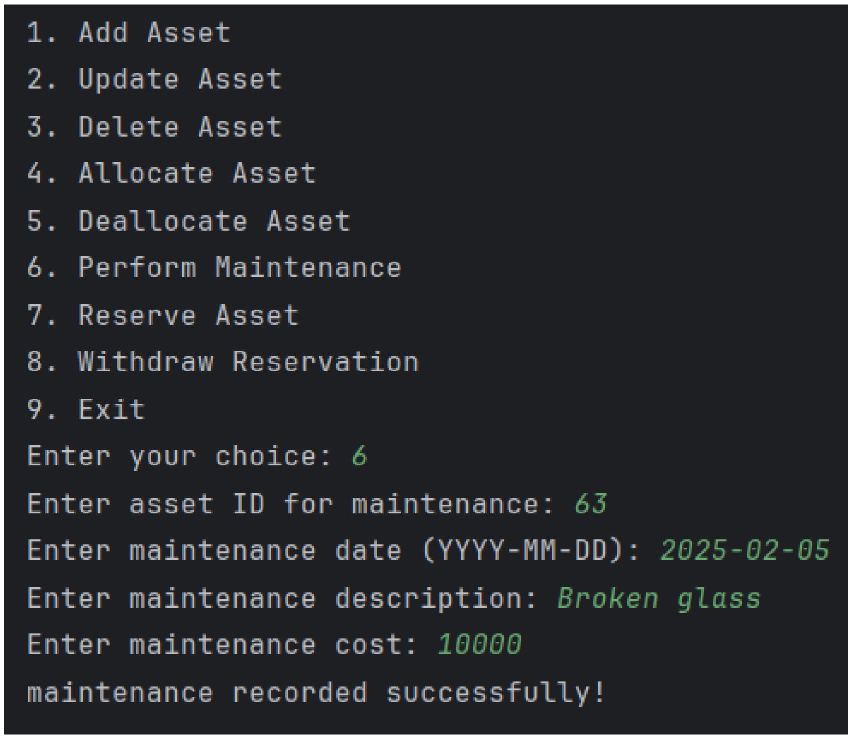
```
1. Add Asset
2. Update Asset
3. Delete Asset
4. Allocate Asset
5. Deallocate Asset
6. Perform Maintenance
7. Reserve Asset
8. Withdraw Reservation
9. Exit
Enter your choice: 4
Enter asset ID to allocate: 63
Enter employee ID: 3
Enter allocation date (YYYY-MM-DD): 2025-02-03
asset 63 allocated successfully to employee 3!
```

5) DEALLOCATE ASSET



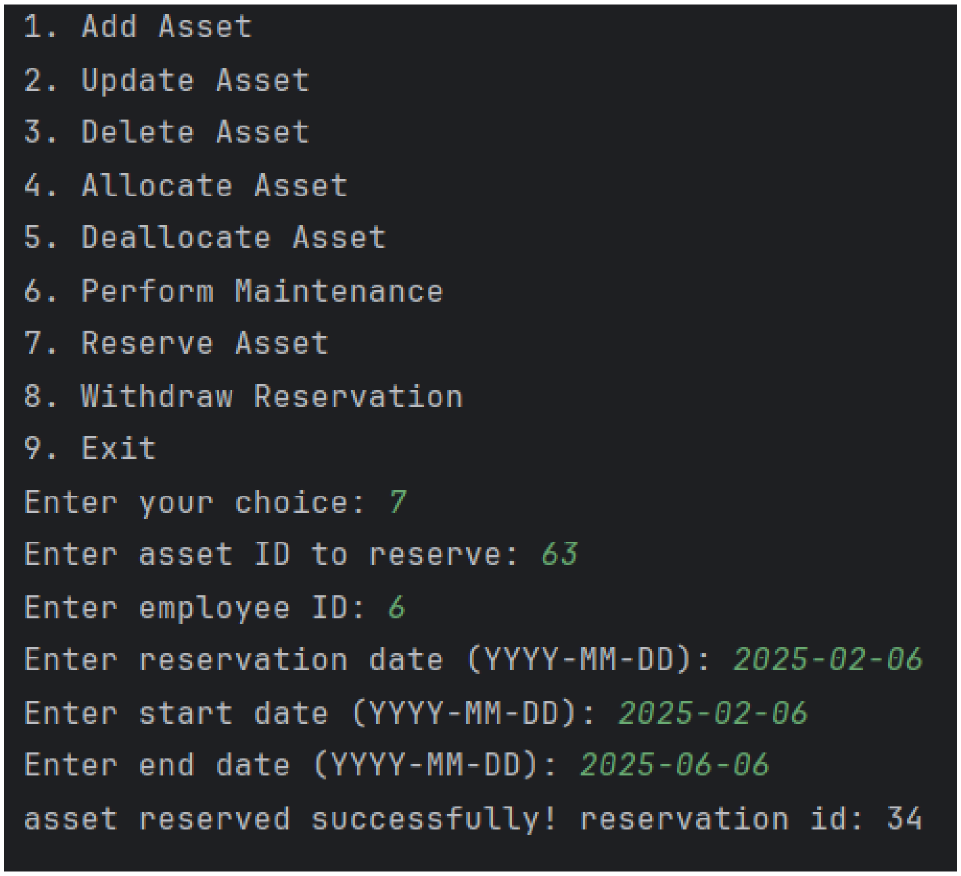
```
1. Add Asset
2. Update Asset
3. Delete Asset
4. Allocate Asset
5. Deallocate Asset
6. Perform Maintenance
7. Reserve Asset
8. Withdraw Reservation
9. Exit
Enter your choice: 5
Enter asset ID to deallocate: 63
asset deallocated successfully!
```

6) PERFORM MAINTENANCE



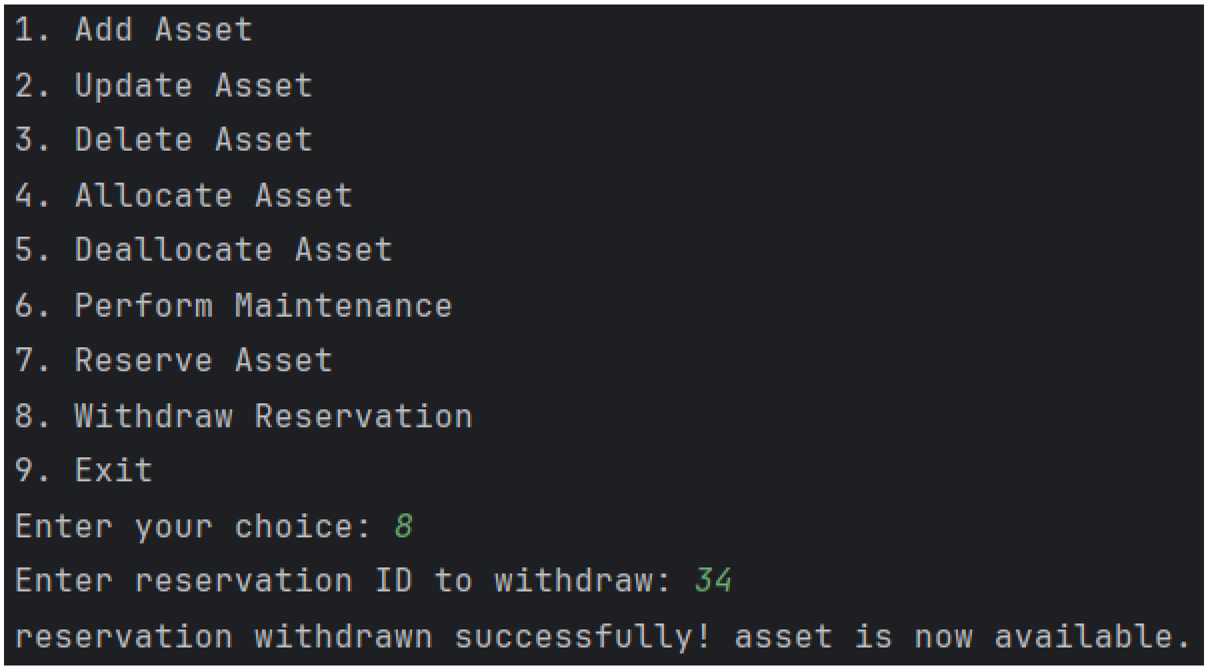
```
1. Add Asset
2. Update Asset
3. Delete Asset
4. Allocate Asset
5. Deallocate Asset
6. Perform Maintenance
7. Reserve Asset
8. Withdraw Reservation
9. Exit
Enter your choice: 6
Enter asset ID for maintenance: 63
Enter maintenance date (YYYY-MM-DD): 2025-02-05
Enter maintenance description: Broken glass
Enter maintenance cost: 10000
maintenance recorded successfully!
```

7) RESERVE ASSET



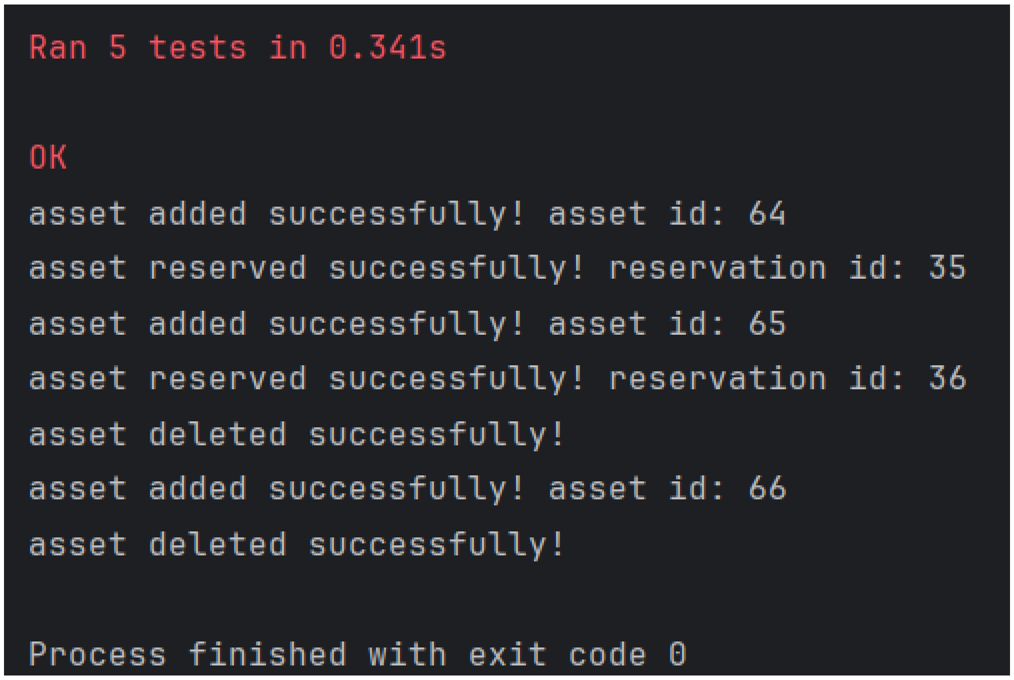
```
1. Add Asset
2. Update Asset
3. Delete Asset
4. Allocate Asset
5. Deallocate Asset
6. Perform Maintenance
7. Reserve Asset
8. Withdraw Reservation
9. Exit
Enter your choice: 7
Enter asset ID to reserve: 63
Enter employee ID: 6
Enter reservation date (YYYY-MM-DD): 2025-02-06
Enter start date (YYYY-MM-DD): 2025-02-06
Enter end date (YYYY-MM-DD): 2025-06-06
asset reserved successfully! reservation id: 34
```

8) WITHDRAW RESERVATION



```
1. Add Asset
2. Update Asset
3. Delete Asset
4. Allocate Asset
5. Deallocate Asset
6. Perform Maintenance
7. Reserve Asset
8. Withdraw Reservation
9. Exit
Enter your choice: 8
Enter reservation ID to withdraw: 34
reservation withdrawn successfully! asset is now available.
```

UNIT TESTING



```
Ran 5 tests in 0.341s

OK
asset added successfully! asset id: 64
asset reserved successfully! reservation id: 35
asset added successfully! asset id: 65
asset reserved successfully! reservation id: 36
asset deleted successfully!
asset added successfully! asset id: 66
asset deleted successfully!

Process finished with exit code 0
```

CONCLUSION

The Digital Asset Management System represents a significant improvement over manual asset tracking methods, providing organizations with accurate, real-time visibility of their physical resources. By combining Python's programming capabilities with MySQL's data management strengths, the system delivers reliable performance while remaining accessible to users. Future enhancements could incorporate mobile access, automated notifications, or integration with procurement systems. As organizations continue to recognize the importance of efficient asset management, systems like this will play an increasingly vital role in operational efficiency and cost control.