**II. Phase: Business Process Modeling (Related to Management Information Systems - MIS)**

**1. Scope Definition**

**Business Process:**

Managing waste collection requests from residents and businesses, scheduling waste pickup, and logging service completion.

**Relevance to MIS:**

This process involves capturing service requests, coordinating resources (e.g., trucks, personnel), and tracking performance metrics, core MIS functions.

**Objective:**

Automate waste collection flow, monitor collector activities, and generate reports for decision-making.

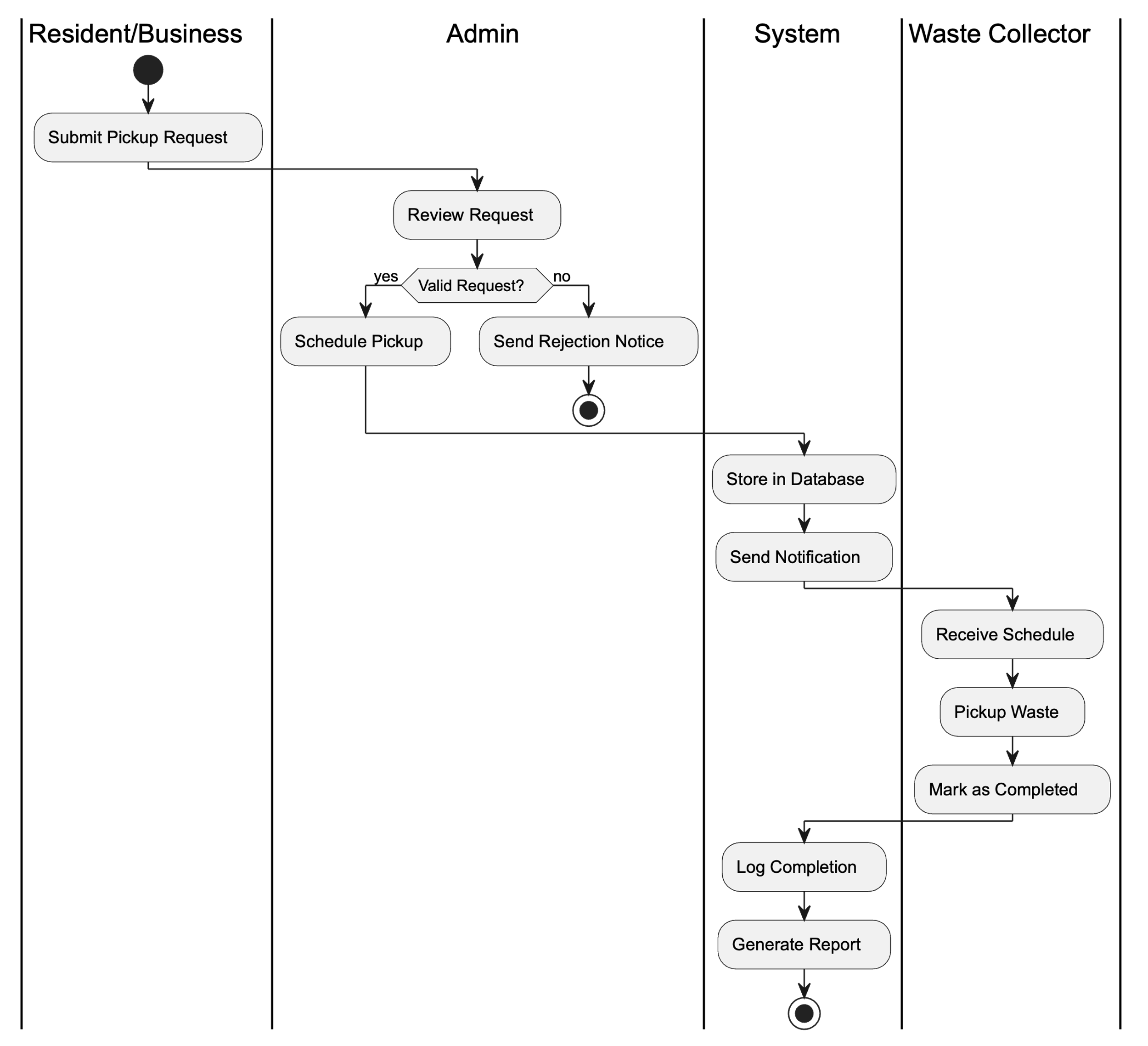
**Expected Outcomes:**

* Faster service response
* Efficient scheduling
* Digital tracking of performance
* Reduced manual errors

**2. Key Entities (Actors and Systems)**

| **Entity** | **Role/Responsibility** |
| --- | --- |
| Resident/Business | Submit waste pickup requests |
| Admin (Municipality) | Reviews, approves, and schedules requests |
| Waste Collector | Fulfills assigned pickups and logs completion |
| System Database | Stores requests, schedules, logs, user info |
| Notification System | Sends reminders and alerts |

**3–5. Diagram with Swimlanes, UML Notation, and Logical Flow**



**6. Brief Explanation (for your report)**

The diagram models the waste collection process starting from the resident’s request to the final report generation. It includes key actors like the Admin, Waste Collector, and the automated system.

This system supports **MIS** by ensuring timely scheduling, digital record-keeping, and real-time reporting. It improves **decision-making** through analytics on request frequency, collector performance, and complaint resolution.

The process enhances **operational efficiency** by reducing manual coordination and delays, ensuring accountability via logs and reporting.

**III. Phase: Logical Model Design**

**1. Entity-Relationship (ER) Model**

In this section, we define the main entities involved in the Waste Management System and their key attributes, data types, and relationships.

**1.1. Identified Entities and Attributes**

| **Entity** | **Attributes** |
| --- | --- |
| **User** | user\_id (PK), name, email, phone, user\_type, location |
| **Waste\_Request** | request\_id (PK), user\_id (FK), request\_date, status |
| **Collector** | collector\_id (PK), name, phone, vehicle\_id (FK) |
| **Vehicle** | vehicle\_id (PK), plate\_number, capacity\_kg, status |
| **Schedule** | schedule\_id (PK), request\_id (FK), collector\_id (FK), scheduled\_date, actual\_collection\_date, status |
| **Complaint** | complaint\_id (PK), user\_id (FK), request\_id (FK), message, complaint\_date, status |

Each attribute is associated with a proper data type (e.g., NUMBER, VARCHAR2, DATE) as per Oracle PL/SQL standards.

**2. Relationships & Constraints**

**2.1 Relationships Between Entities**

* **User – Waste\_Request**: One user can submit many requests (1-to-many).
* **Waste\_Request – Schedule**: One request has one schedule (1-to-1).
* **Collector – Schedule**: One collector can be assigned to multiple schedules (1-to-many).
* **Collector – Vehicle**: One collector uses one vehicle (1-to-1).
* **Waste\_Request – Complaint**: A request may have many complaints (1-to-many).
* **User – Complaint**: A user can file multiple complaints (1-to-many).

**2.2 Constraints Applied**

* **Primary Keys**: All tables have defined primary keys to uniquely identify each record.
* **Foreign Keys**: Relationships are enforced using foreign key constraints.
* **NOT NULL**: Applied to mandatory fields like user\_id, request\_date, etc.
* **UNIQUE**: Enforced on fields like email, plate\_number.
* **CHECK**: Applied to fields like status and user\_type to ensure valid values.
* **DEFAULT**: Used to assign initial values to fields like status.

**3. Normalization**

All database tables are designed to follow the **Third Normal Form (3NF)**:

* **1NF**: All attributes hold atomic values.
* **2NF**: All non-key attributes are fully dependent on the primary key.
* **3NF**: No transitive dependencies exist between non-key attributes.

This ensures **data integrity, reduced redundancy**, and better **scalability**.

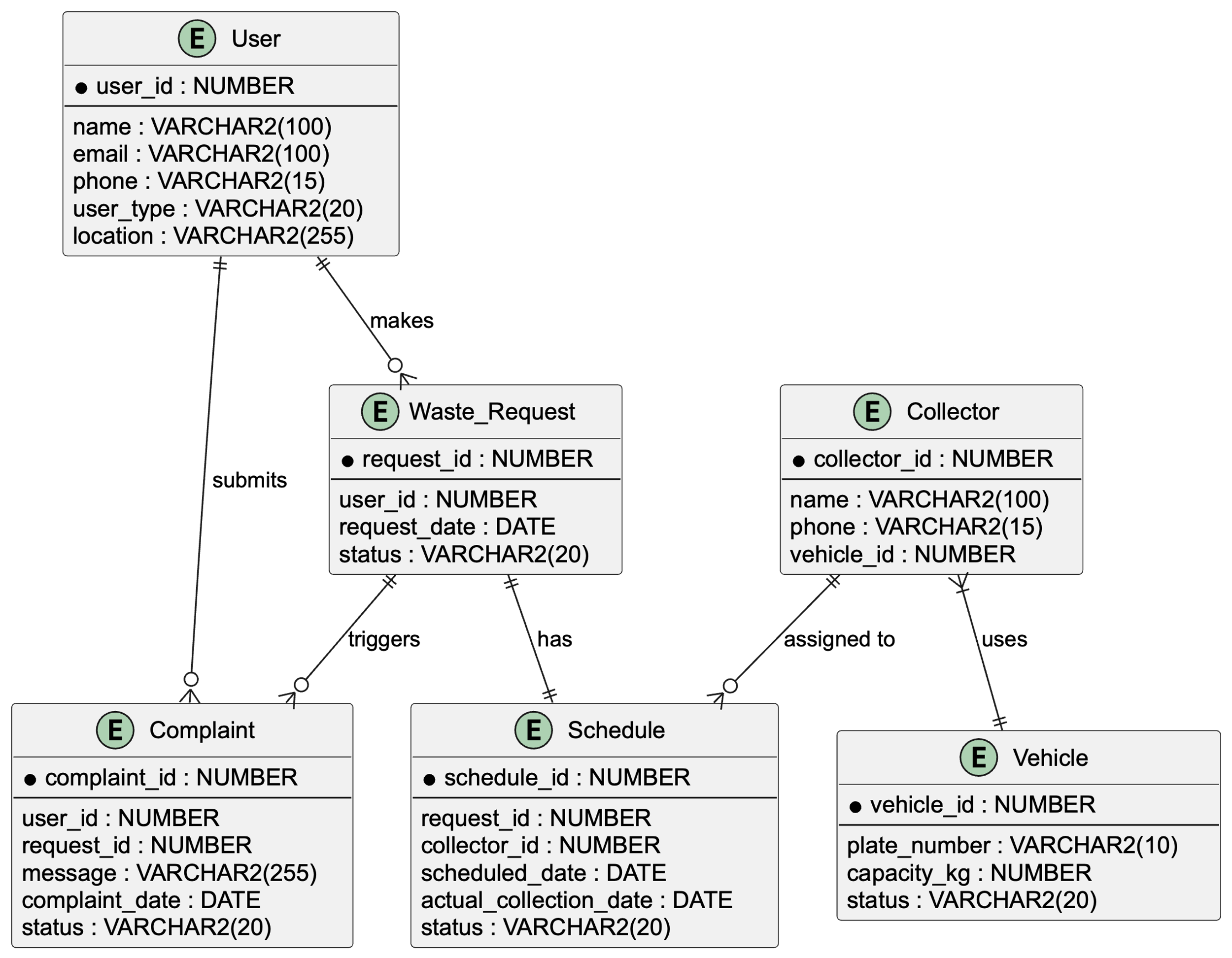
**4. Handling Data Scenarios**

The logical model supports a wide range of realistic scenarios, including:

* Users submitting multiple requests across different dates and locations.
* Assigning collectors to overlapping schedules.
* Allowing users to file complaints against specific requests.
* Vehicle reuse tracking and capacity monitoring.
* Tracking request statuses through multiple stages (Pending, Approved, Completed, etc.)

**5. ER Diagram**

Below is the ER diagram for the Waste Management System, modeled using Draw.io:



**IV. Phase: Database (Pluggable Database) Creation and Naming**

**1. Database Creation in Oracle**

**🧾 Naming Convention**

Your pluggable database should be named as:

mon\_27901\_vanessa\_wastemgt\_db

**🔐 Password**

Use your **first name** as the password:

Password: vanessa

**📜 SQL Script to Create the Database User and Assign Privileges**

Here’s a basic SQL block you can execute in Oracle SQL\*Plus or SQL Developer:

-- Create user

CREATE USER mon\_27901\_vanessa\_wastemgt\_db

IDENTIFIED BY jeandd;

-- Grant DBA privileges (super admin)

GRANT CONNECT, RESOURCE, DBA TO CREATE USER mon\_27901\_vanessa\_wastemgt\_db

-- Optionally, set default tablespace and quota (recommended)

ALTER USER mon\_27901\_vanessa\_wastemgt\_db DEFAULT TABLESPACE USERS;

ALTER USER mon\_27901\_vanessa\_wastemgt\_db QUOTA UNLIMITED ON USERS;

⚠️ Make sure you are logged in as a user with **ADMIN privileges** (e.g., SYSDBA) to run these commands.

**2. Oracle Enterprise Manager (OEM) Setup**

You need to:

* Log in to **Oracle Enterprise Manager**.
* Connect to the **PDB (Pluggable Database)** you created.
* Monitor the following:
  + Tablespace usage
  + Session statistics
  + SQL activity (like DML and DDL)

**📸 Screenshot Tips for GitHub Submission**

When you reach this step, capture screenshots that clearly show:

* Your **database name** (mon\_27901\_vanessa\_wastemgt\_db)
* The **OEM dashboard overview**
* Any **SQL activity logs**, session reports, or metrics
* **User creation confirmation**

Make sure the **screenshots are clean and readable** before pushing them to GitHub.

**✅ Summary for Report**

You can write this in your report under **Phase IV**:

**Phase IV: Database Creation and Monitoring**

As part of this phase, a pluggable database namedmon\_27901\_vanessa\_wastemgt\_db was created using Oracle. A privileged user was defined with the username matching the database name and the password set to the student's first name for compliance with the naming convention.

To manage and monitor the database, Oracle Enterprise Manager (OEM) was configured. It tracks database health, session activity, resource usage, and query logs. Screenshots from OEM are provided in the GitHub repository to verify setup and user activity.

The user was granted full DBA privileges to allow creation and testing of PL/SQL components such as procedures, functions, triggers, and packages in later phases.

**V: Physical Database Implementation**

**1. Table Creation – SQL DDL Statements**

Here's a full PL/SQL script to create all the necessary tables with proper data types, primary/foreign keys, and constraints:

-- USER table

CREATE TABLE Users (

user\_id NUMBER PRIMARY KEY,

name VARCHAR2(100) NOT NULL,

email VARCHAR2(100) UNIQUE NOT NULL,

phone VARCHAR2(15),

user\_type VARCHAR2(20) CHECK (user\_type IN ('Resident', 'Business', 'Admin')),

location VARCHAR2(255)

);

-- VEHICLE table

CREATE TABLE Vehicles (

vehicle\_id NUMBER PRIMARY KEY,

plate\_number VARCHAR2(10) UNIQUE NOT NULL,

capacity\_kg NUMBER NOT NULL,

status VARCHAR2(20) DEFAULT 'Active'

);

-- COLLECTOR table

CREATE TABLE Collectors (

collector\_id NUMBER PRIMARY KEY,

name VARCHAR2(100) NOT NULL,

phone VARCHAR2(15),

vehicle\_id NUMBER UNIQUE,

CONSTRAINT fk\_vehicle FOREIGN KEY (vehicle\_id) REFERENCES Vehicles(vehicle\_id)

);

-- WASTE\_REQUEST table

CREATE TABLE Waste\_Requests (

request\_id NUMBER PRIMARY KEY,

user\_id NUMBER NOT NULL,

request\_date DATE NOT NULL,

status VARCHAR2(20) CHECK (status IN ('Pending', 'Approved', 'Rejected', 'Completed')),

CONSTRAINT fk\_user FOREIGN KEY (user\_id) REFERENCES Users(user\_id)

);

-- SCHEDULE table

CREATE TABLE Schedules (

schedule\_id NUMBER PRIMARY KEY,

request\_id NUMBER UNIQUE NOT NULL,

collector\_id NUMBER NOT NULL,

scheduled\_date DATE,

actual\_collection\_date DATE,

status VARCHAR2(20) CHECK (status IN ('Scheduled', 'Missed', 'Collected')),

CONSTRAINT fk\_request FOREIGN KEY (request\_id) REFERENCES Waste\_Requests(request\_id),

CONSTRAINT fk\_collector FOREIGN KEY (collector\_id) REFERENCES Collectors(collector\_id)

);

-- COMPLAINT table

CREATE TABLE Complaints (

complaint\_id NUMBER PRIMARY KEY,

user\_id NUMBER NOT NULL,

request\_id NUMBER NOT NULL,

message VARCHAR2(255),

complaint\_date DATE DEFAULT SYSDATE,

status VARCHAR2(20) DEFAULT 'Open',

CONSTRAINT fk\_comp\_user FOREIGN KEY (user\_id) REFERENCES Users(user\_id),

CONSTRAINT fk\_comp\_request FOREIGN KEY (request\_id) REFERENCES Waste\_Requests(request\_id)

);

2. Sample Data Insertion – SQL DML Statements

-- Users

INSERT INTO Users VALUES (1, 'Alice Johnson', 'alice@mail.com', '078000001', 'Resident', 'Sector 1');

INSERT INTO Users VALUES (2, 'GreenTech Ltd', 'info@greentech.com', '078000002', 'Business', 'Industrial Zone');

-- Vehicles

INSERT INTO Vehicles VALUES (1, 'RAB123A', 2000, 'Active');

INSERT INTO Vehicles VALUES (2, 'RAC567B', 1500, 'Active');

-- Collectors

INSERT INTO Collectors VALUES (1, 'John Doe', '0781111111', 1);

INSERT INTO Collectors VALUES (2, 'Jane Smith', '0782222222', 2);

-- Waste Requests

INSERT INTO Waste\_Requests VALUES (101, 1, TO\_DATE('2025-05-01', 'YYYY-MM-DD'), 'Pending');

INSERT INTO Waste\_Requests VALUES (102, 2, TO\_DATE('2025-05-02', 'YYYY-MM-DD'), 'Approved');

-- Schedules

INSERT INTO Schedules VALUES (1001, 101, 1, TO\_DATE('2025-05-03', 'YYYY-MM-DD'), NULL, 'Scheduled');

INSERT INTO Schedules VALUES (1002, 102, 2, TO\_DATE('2025-05-04', 'YYYY-MM-DD'), TO\_DATE('2025-05-04', 'YYYY-MM-DD'), 'Collected');

-- Complaints

INSERT INTO Complaints VALUES (5001, 1, 101, 'Trash was not picked on time', TO\_DATE('2025-05-04', 'YYYY-MM-DD'), 'Open');

**3. Data Integrity Measures Applied**

* **Primary keys** ensure entity uniqueness.
* **Foreign keys** enforce relational integrity.
* **UNIQUE** and **CHECK** constraints ensure valid and non-duplicate values.
* **DEFAULT** values prevent NULLs where optional data applies.

All inserted data reflects real-world use cases: missed pickups, scheduled collections, valid email formats, and meaningful complaint messages.

**Phase V: Table Implementation and Data Insertion**

In this phase, the logical model was translated into a physical Oracle database structure. Six tables were created: Users, Vehicles, Collectors, Waste\_Requests, Schedules, and Complaints.

Each table includes appropriate data types, constraints, and relationships. Sample data was inserted to support real-world testing and to validate system operations, such as scheduling pickups, assigning collectors, and recording complaints.

Constraints such as NOT NULL, UNIQUE, CHECK, and DEFAULT were applied to ensure data integrity and consistency. The design supports practical scenarios like rejected requests, pending schedules, and complaint tracking, ensuring reliability for all project use cases.

**VI: Database Interaction and Transactions**

**1. DML and DDL Operations**

**✅ DDL Example – Add New Column**

ALTER TABLE Waste\_Requests ADD description VARCHAR2(255);

**✅ DML Examples**

-- INSERT

INSERT INTO Waste\_Requests VALUES (103, 1, SYSDATE, 'Pending', 'Extra pickup needed');

-- UPDATE

UPDATE Waste\_Requests SET status = 'Completed' WHERE request\_id = 101;

-- DELETE

DELETE FROM Complaints WHERE complaint\_id = 5001;

**2. Problem Statement for Analytics Task**

“Determine the total number of waste pickups performed per collector per month, and find the collector with the highest number of collections.”

This will be used in a **function** using **window functions**.

**3. Stored Procedure: Fetch User Complaints**

CREATE OR REPLACE PROCEDURE GetComplaintsByUser (

p\_user\_id IN NUMBER

)

IS

CURSOR comp\_cursor IS

SELECT complaint\_id, message, status

FROM Complaints

WHERE user\_id = p\_user\_id;

v\_id Complaints.complaint\_id%TYPE;

v\_msg Complaints.message%TYPE;

v\_status Complaints.status%TYPE;

BEGIN

OPEN comp\_cursor;

LOOP

FETCH comp\_cursor INTO v\_id, v\_msg, v\_status;

EXIT WHEN comp\_cursor%NOTFOUND;

DBMS\_OUTPUT.PUT\_LINE('Complaint ID: ' || v\_id || ', Message: ' || v\_msg || ', Status: ' || v\_status);

END LOOP;

CLOSE comp\_cursor;

EXCEPTION

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE('Error occurred: ' || SQLERRM);

END;

✅ Use EXEC GetComplaintsByUser(1); to test it.

**4. Function with Window Function: Top Collector by Count**

CREATE OR REPLACE FUNCTION TopCollector RETURN VARCHAR2 IS

v\_name VARCHAR2(100);

BEGIN

SELECT name INTO v\_name

FROM (

SELECT c.name, COUNT(\*) AS total,

RANK() OVER (ORDER BY COUNT(\*) DESC) AS rk

FROM Collectors c

JOIN Schedules s ON c.collector\_id = s.collector\_id

GROUP BY c.name

) WHERE rk = 1;

RETURN v\_name;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

RETURN 'No data';

WHEN OTHERS THEN

RETURN 'Error: ' || SQLERRM;

END;

✅ Use SELECT TopCollector FROM dual; to call it.

**5. Package Example: Waste\_Analytics\_PKG**

CREATE OR REPLACE PACKAGE Waste\_Analytics\_PKG AS

PROCEDURE PrintAllComplaints;

FUNCTION CountRequestsByUser(p\_user\_id NUMBER) RETURN NUMBER;

END Waste\_Analytics\_PKG;

/

CREATE OR REPLACE PACKAGE BODY Waste\_Analytics\_PKG AS

PROCEDURE PrintAllComplaints IS

CURSOR cur IS SELECT user\_id, message FROM Complaints;

v\_user Complaints.user\_id%TYPE;

v\_msg Complaints.message%TYPE;

BEGIN

FOR rec IN cur LOOP

DBMS\_OUTPUT.PUT\_LINE('User: ' || rec.user\_id || ' | Message: ' || rec.message);

END LOOP;

END;

FUNCTION CountRequestsByUser(p\_user\_id NUMBER) RETURN NUMBER IS

v\_count NUMBER;

BEGIN

SELECT COUNT(\*) INTO v\_count

FROM Waste\_Requests

WHERE user\_id = p\_user\_id;

RETURN v\_count;

EXCEPTION

WHEN OTHERS THEN

RETURN -1;

END;

END Waste\_Analytics\_PKG;

✅ Test with:

EXEC Waste\_Analytics\_PKG.PrintAllComplaints;

SELECT Waste\_Analytics\_PKG.CountRequestsByUser(1) FROM dual;

**✅ Summary for Report**

In this phase, core interactions with the database were developed using PL/SQL. This includes:

- Executing DML (Insert, Update, Delete) and DDL (Alter) commands.

- Creating reusable procedures with cursor loops to fetch user complaint data.

- Writing analytical functions with window functions to retrieve top-performing collectors.

- Encapsulating logic into a package `Waste\_Analytics\_PKG` that groups related procedures and functions.

- Implementing error handling and exception capture for robustness.

The modular approach ensures clean code organization and easier maintenance.

**VII: Advanced Database Programming and Auditing**

**1. Problem Statement**

In our Waste Management System, we want to **enhance data integrity and security** by preventing unauthorized table modifications during **weekdays (Mon–Fri)** and **public holidays**. We also aim to **track all critical DML operations** (INSERT, UPDATE, DELETE) for accountability.

**Justification**

* Triggers help us **enforce business rules** at the database level.
* Auditing mechanisms allow tracking **who did what and when**.
* A holiday reference table ensures the system respects operational calendars.

**2. Reference Table for Holidays**

CREATE TABLE Holidays (

holiday\_date DATE PRIMARY KEY,

description VARCHAR2(100)

);

-- Sample data (upcoming month)

INSERT INTO Holidays VALUES (TO\_DATE('2025-06-01', 'YYYY-MM-DD'), 'Independence Day');

INSERT INTO Holidays VALUES (TO\_DATE('2025-06-16', 'YYYY-MM-DD'), 'National Cleanup Day');

**3. Auditing Table**

CREATE TABLE Audit\_Log (

log\_id NUMBER GENERATED ALWAYS AS IDENTITY PRIMARY KEY,

user\_id NUMBER,

action\_date DATE DEFAULT SYSDATE,

operation\_type VARCHAR2(10),

object\_name VARCHAR2(50),

status VARCHAR2(10)

);

**4. Security Trigger (Preventing DML on Weekdays and Holidays)**

CREATE OR REPLACE TRIGGER trg\_block\_dml

BEFORE INSERT OR UPDATE OR DELETE ON Waste\_Requests

DECLARE

v\_day VARCHAR2(10);

v\_today DATE := SYSDATE;

v\_holiday NUMBER;

BEGIN

SELECT TO\_CHAR(v\_today, 'DY') INTO v\_day FROM dual;

SELECT COUNT(\*) INTO v\_holiday

FROM Holidays

WHERE holiday\_date = TRUNC(v\_today);

IF v\_day IN ('MON', 'TUE', 'WED', 'THU', 'FRI') OR v\_holiday > 0 THEN

RAISE\_APPLICATION\_ERROR(-20001, 'DML operations not allowed today.');

END IF;

END;

/

**5. Compound Trigger for Auditing**

CREATE OR REPLACE TRIGGER trg\_audit\_waste\_requests

AFTER INSERT OR UPDATE OR DELETE ON Waste\_Requests

FOR EACH ROW

BEGIN

INSERT INTO Audit\_Log (user\_id, operation\_type, object\_name, status)

VALUES (USERENV('SESSIONID'), CASE

WHEN INSERTING THEN 'INSERT'

WHEN UPDATING THEN 'UPDATE'

WHEN DELETING THEN 'DELETE'

END,

'Waste\_Requests',

'allowed');

END;

/

**✅ Summary for Your Report**

This phase implemented advanced PL/SQL programming to automate operations and enforce business rules. A holiday table and trigger ensure no DML activity occurs during weekdays or national holidays. A compound trigger logs every change to the `Waste\_Requests` table into an `Audit\_Log`, capturing the user session, action, and timestamp.

These mechanisms enhance security, enable accountability, and align with MIS best practices for database-level enforcement and monitoring.

**GitHub Report Structure (Markdown Example)**

Create a README.md file in your GitHub repo with the following structure:

# Waste Management System – Capstone Project

\*\*Course\*\*: INSY 8311 - Database Development with PL/SQL

\*\*Student\*\*: HABIMANA Jean de Dieu

\*\*Student ID\*\*: 27901

## 🧠 Problem Statement

Urban areas lack centralized tools for monitoring waste pickups, complaints, and service efficiency. This project builds a PL/SQL-based Oracle database to automate workflows and improve operational control.

## 📊 Key Entities & Relationships

- Users, Waste Requests, Schedules, Vehicles, Collectors, Complaints

- Normalized to 3NF

- Includes constraints and auditing logic

## 🧱 SQL Scripts

Organized into phases:

- `phase\_i\_ii\_overview.md`: Problem, Process Modeling

- `phase\_iii\_logical\_model.puml`: ER Diagram

- `phase\_v\_tables\_insertion.sql`

- `phase\_vi\_interactions.sql`

- `phase\_vii\_advanced\_auditing.sql`

## 📸 Screenshots

- OEM setup

- Sample query results

- Trigger & audit logs

## ✅ Features

- PL/SQL Triggers to prevent actions on weekdays/holidays

- Audit table for logging operations

- Reusable packages and cursor-based procedures

## 🚀 Recommendations

- Add GIS integration

- Implement mobile collector tracking

- Use AI for smart scheduling

---

📂 Be sure to include `.sql`, `.puml`, and screenshots inside properly named folders (e.g., `/scripts`, `/diagrams`, `/screenshots`).

---

Let me know if you'd like help generating the `README.md` file or screenshots. You're now fully ready to complete Phase VIII and submit your capstone! ​:contentReference[oaicite:0]{index=0}​