DAMG 7250 Project Report Hamilton Street Railway Master Database

Group 3 Team Members

Name	NUID
KARAN PARMAR	002272304
SHRAJNA SHETTY	002693529
ABDUL MOHSIN AHMED	002818634
MOHAMMED ZAHEERUDDIN AKHTAR SIDDIQUI	002832558
SAAHIL KARNIK	002209455
AMAN KHAN	002299467

Supervisor – Prof. Syed Farhan Mazhar

MSIS, NeU Fall 2024

Project - "Hamilton Street Railway Master Database"

1. Purpose of the Project

Hamilton Street Railway (HSR) is a public transport agency providing bus service in the Hamilton area. Our team plans to design a database that tracks the activity and operations of its bus fleet so they can analyze the utilization of buses and its resources in serving customers. The purpose of this project is to provide a dexterous, scalable, and secured database solution that will help achieve business requirements.

2. Project Outline

Phases	Operations	Timeline
Phase 1: Requirement Gathering	Conduct meetings with stakeholders and customers to understand data requirements.	7 Days
Phase 2: Conceptual Modelling	Define entities such as Persons, Stops, Routes, and their attributes	2-3 Days
Phase 3: Logical Modelling	Translate the conceptual model into a detailed logical model. Specify data types, primary and foreign keys, and normalization.	2-3 Days
Phase 4: Physical Database Design	Create tables, establish relationships, and implement indexing for performance.	4-5 Days
Phase 5: Data Population	Populate the database with sample data for testing and validation. Ensure data accuracy and completeness.	5-7 Days
Phase 6: Documentation	Document the database schema, relationships, and any custom queries.	2-3 Days

3. Project Objectives and Outcomes

We aim to meet a few basic but important objectives for the Hamilton Street Railway through this project. They are -

Data Collection and Integration: Collect, validate, and integrate various transport data sources to create a comprehensive and unified database for timely and accurate information.
Data Quality Assurance: Implement data quality control measures to ensure the accuracy, reliability, and consistency of the data, minimizing errors and redundancies.
Data Analysis and Reporting: Support data analysis and reporting capabilities to derive insights, identify trends, and inform evidence-based decision-making in route and resource management.
Cost Efficiency: Optimise resource utilization to ensure the cost-effective operation and maintenance of the database system.
Disaster Recovery and Data Backup: Establish a robust disaster recovery and data backup plan to protect passenger and driver data from loss or corruption.

The project outcome of this database design will yield several positive results, contributing to overall business success. These include -

Improved Data Accessibility: Users will have a panoramic view of data and affiliations, fostering better decision-making and strategic planning.
Efficient Operations: Streamlined data management of Customer, Dealer, andVehicle data will lead to more efficient and organized business processes.
Enhanced Customer Satisfaction: The focus on enhanced customer experience through better business decisions will likely result in improved satisfaction levels, leading to customer retention and loyalty.
Effective Business Intelligence: The establishment of data warehousing facilitates robust business intelligence, enabling insightful analysis and reporting.

4. Business Case

- **Problem**

  Irrespective of the state-of-the-art advanced systems used by Hamilton Street Railways to carry out the day-to-day transportation business, HSR doesn't have a unified database for customer-centric and bus-centric information. So, if the organization wants to visualize all the routes, stops and buses available or passenger and driver interaction, then it would be a cumbersome manual process. To that end, the following database would provide an efficacious system which would help the organization to view all the transport-related data.

- **Solution**

  The HSR Master System database would provide the ability to view the four core components in the organization (i.e.) Persons (and their respective child elements), Buses, Routes, and Stops. The robust intelligent system functions as a source of truth for a large variety of reports for stakeholders using which a lot of important decisions can be taken and innovative investments can be made.

- **Benefits**

  Master Database – functions as the one master system for customer and resource information which would ingest data from all the sources and provide the standardized information for downstream systems.

- Customizable views – the system provides bite sized views like Person, Bus, Route, Stop etc. This aids in focusing on key elements of the entire data at a time and helps in ease of comprehension and perspective.
- Reduce data redundancy – This database eliminates the need for

multiple discrete databases for customers, resource and route information and unwanted normalization or synchronization between them.

- Informed Decision-making – This system aids in generating a large variety of projections on sales and service which helps the stakeholders in making profitable business decisions.

5. Tools & Techniques

- **Collecting Requirement:** We collected requirements from both customers and stakeholders through physical meetings, virtual sessions on Teams, andonline surveys. Additionally, we conducted offline surveys using pen and paper.
- **Conceptual Modelling:** We have created an abstract representation of thedata and its relationships within a system using pen and paper, to understand the essential entities, their attributes, and the relationships between them.
- **Logical Modelling:** A detailed representation that can be implemented in a database management system (DBMS) is made, which defines the structureof the data, including tables, columns, primary and foreign keys, and other constraints.
- **Physical Database Design:** ER diagram is created using Draw.io tool. Normalization, indexing, and performance monitoring for efficiency and reliability techniques are used for creating efficient and reliable databases.

Then, the database is implemented using Oracle SQL developer.

6. Conceptual Data Model

Schema –

The following is a breakdown of our Entity Relationship Diagram-

1. Person: Created to establish Person entity and its attributes within the HSR.

- PersonID (Primary Key)
- FirstName
- LastName

- Gender
- DateOfBirth
- StreetAddress
- CityAddress
- Province
- Occupation

Multiplicity: One-to-One (1:1) with each passenger associated with one person.
Multiplicity: One-to-One (1:1) with each maintenance personnel associated with one person.
Multiplicity: One-to-One (1:1) with each driver associated with one person.

2. Passenger: A type of person who uses the HSR service – passenger, bus driver, maintenance personnel.
- PassengerID (Primary Key, Foreign Key to Person)
- Type (Child, Senior, Student)
- TypeID

Multiplicity: One-to-One (1:1) with each passenger associated with one person.
Multiplicity: One-to-One (1:1) with each passenger associated with one passenger type.

3. PassengerType: Type of passenger – child, senior, student.
- TypeID (Primary Key, Foreign Key to Passenger)
- TypeName
- Amount

Multiplicity: One-to-Many (1:N) - One fare can be associated with many passenger.

4. Bus Driver: Person of type - bus driver.
- DriverID (Primary Key, Foreign Key to Person)
- Salary
- YearsOfService

Multiplicity: One-to-One (1:1) with each bus driver associated with one person.

Multiplicity: One-to-Many (1:N) with each bus driver associated with many driving infractions.

5. Driving Infraction:
- InfractionID (Primary Key)
- DriverID (Foreign Key to Bus Driver)
- Date
- TypeOfInfraction
- DemeritPoints
- FinancialPenalty

Multiplicity: One-to-Many (1:N) - One driver can have multiple driving infractions.

6. Maintenance Personnel: Person of type – Maintenance personnel.
- MaintenanceID (Primary Key, Foreign Key to Person)
- BusID
- AreaSpecialization
- Level
- YearsOfService
- Salary

Multiplicity: One-to-One (1:1) with each maintenance personnel associated with one person.
Multiplicity: One-to-One (1:M) with one bus associated with multiple maintenance personnel.

7. Bus: Created to establish Bus entity and its attributes within the HSR.
- BusID (Primary Key, Foreign key to Maintenance Personnel)
- YearsInOperation
- NumberOfSeats
- Manufacturer
- AdvertisingRevenue
- FuelType
- RouteID

Multiplicity: One-to-Many (1:M) with each Bus associated with many maintenance personnel.

Multiplicity: One-to-Many (1:M) with each Bus associated with many schedules.
Multiplicity: One-to-Many (1:M) with each route associated with many Buses.

8. Route: Created to establish Route entity and its attributes within the HSR.
- RouteID (Primary Key, foreign key to Bus, foreign key to Stop)
- Name

Multiplicity: One-to-Many (1:M) with each route having many route stops
Multiplicity: One-to-Many (1:M) with each route having many buses
Multiplicity: One-to-Many (1:M) with each route having many schedules
Multiplicity: One-to-Many (1:M) with each route having many route sites

9. Stop: Created to establish Stop entity and its attributes within the HSR.
- StopID (Primary Key)
- Name
-RouteID

Multiplicity: One-to-Many (1:M) with each stop having many route stops
Multiplicity: One-to-Many (1:M) with each stop having many schedules

10. Schedule: Created to establish Schedule entity and its attributes within the HSR.
- ScheduleID (Primary Key)
- BusID (Foreign Key to Bus)
- StopID (Foreign Key to Stop)
- RouteID (Foreign Key to Route)
- ArrivalTime
- Date

Multiplicity: Many-to-One (N:1) - Many schedules can belong to one bus, one stop, and one route.

11. Temporary Service: Created to show that in case of route maintenance or any emergency, there will be closure on specific route.
- Service Name (Primary Key)

- Service ID
- Bus ID (Foreign key to Bus)

Multiplicity: One-to-One (1:M) with each service having one route.
Multiplicity: One-to-Many (1:M) with each service having many buses.

12. Event: Created to establish Event entity and its attributes within the HSR.
- EventID (Primary Key)
- SiteID (Foreign Key to Site)
- Name
- DateTime
- ExpectedNumberOfParticipants

Multiplicity: One-to-One (1:1) – One event can be associated with one route.

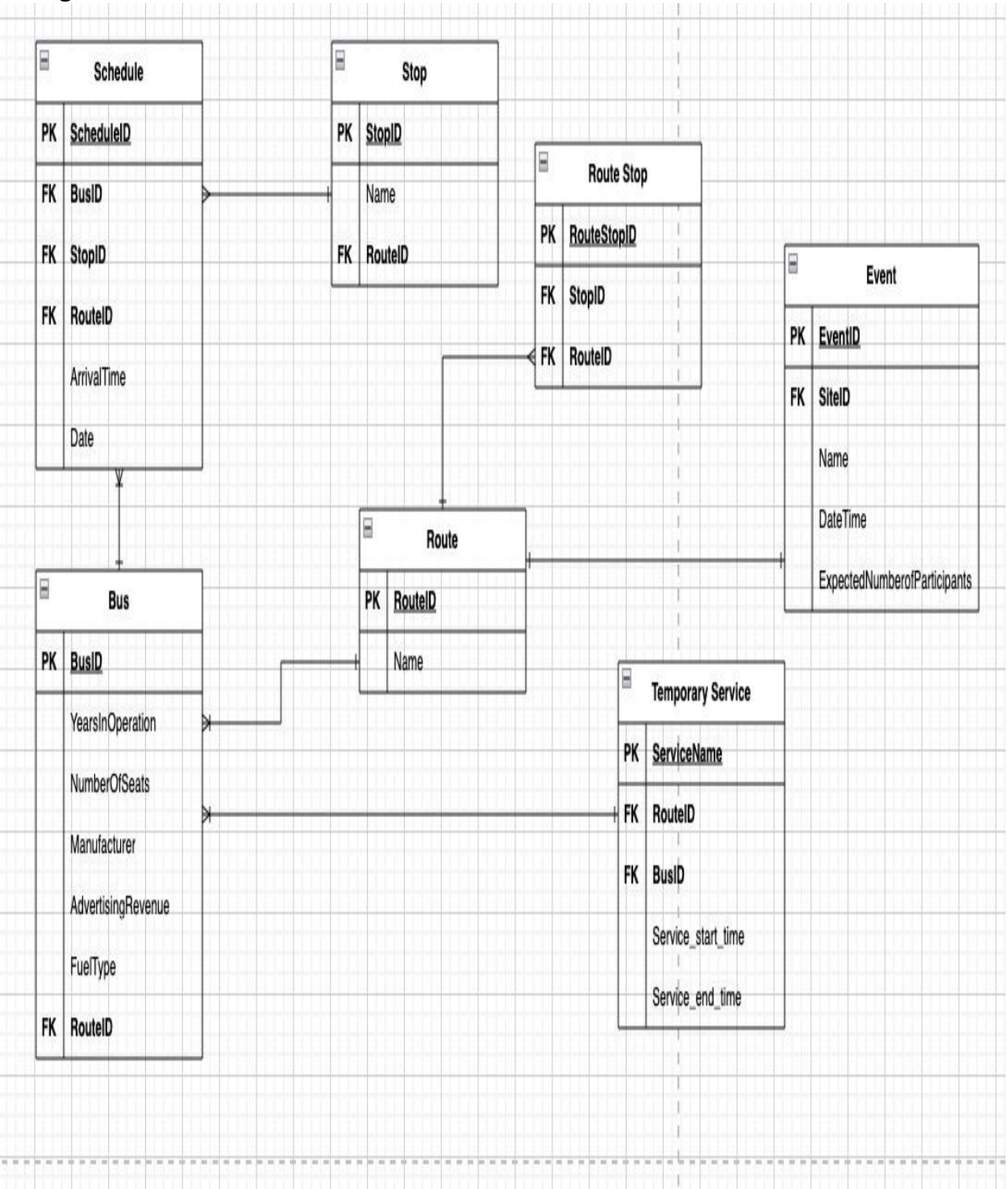
13. RouteStop: Created to handle many to many relationship between route and stop.
-RouteStopID (Primary key)
- RouteID (Foreign Key to Route)
- StopID (Foreign Key to Stop)

Multiplicity: One-to-Many (1:N) – 1 route can have many routestop and many routestopID can have 1 stop

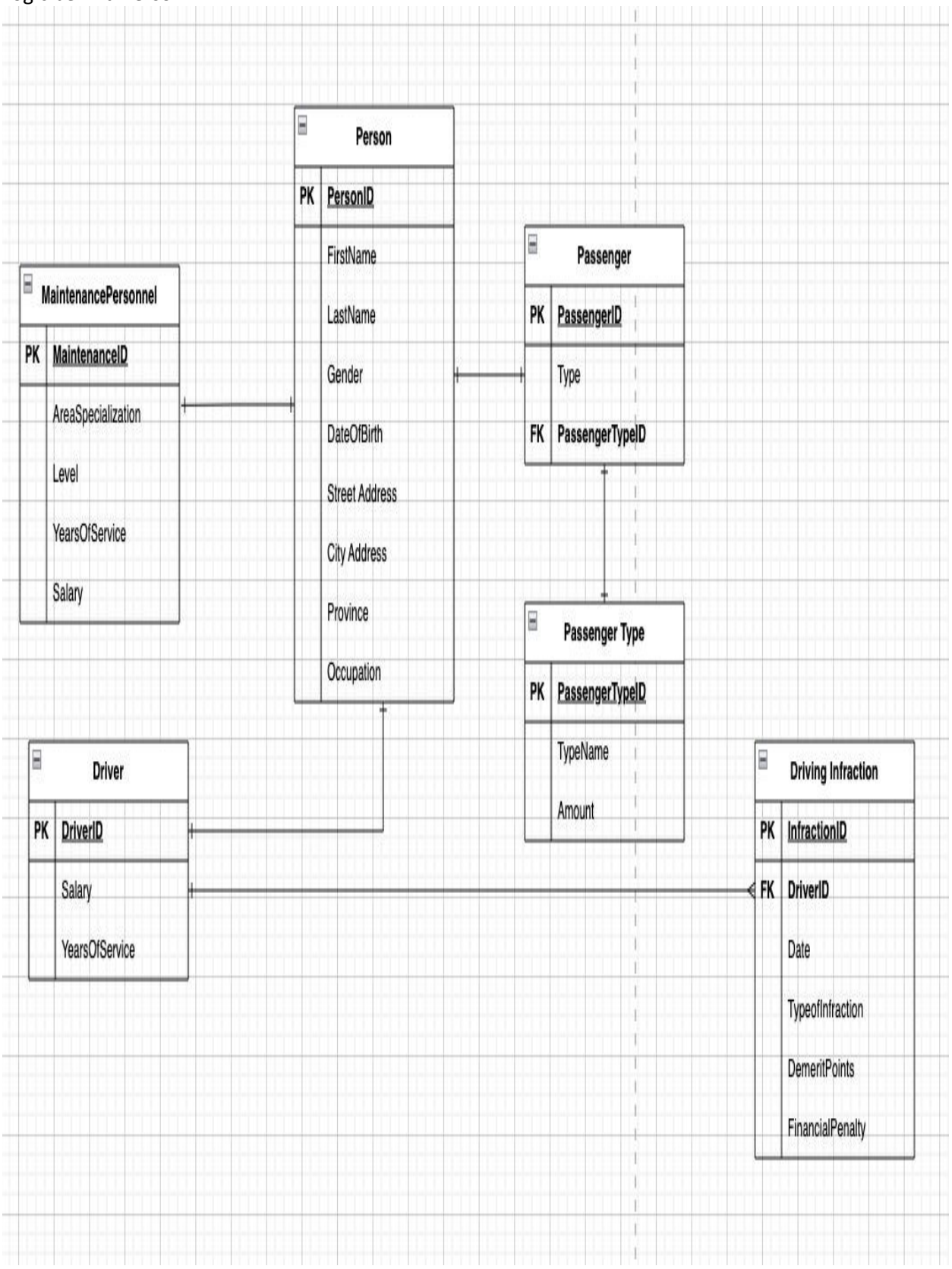
Route Table: Each row represents a unique route.
Stop Table: Each row represents a unique stop.
RouteStop Table (Associative Entity): This table links routes and stops. Each row represents a combination of a route and a stop, indicating that a particular route includes a specific stop.
This way, we've broken down the many-to-many relationship between Route and Stop into two one-to-many relationships.

7. Logical design

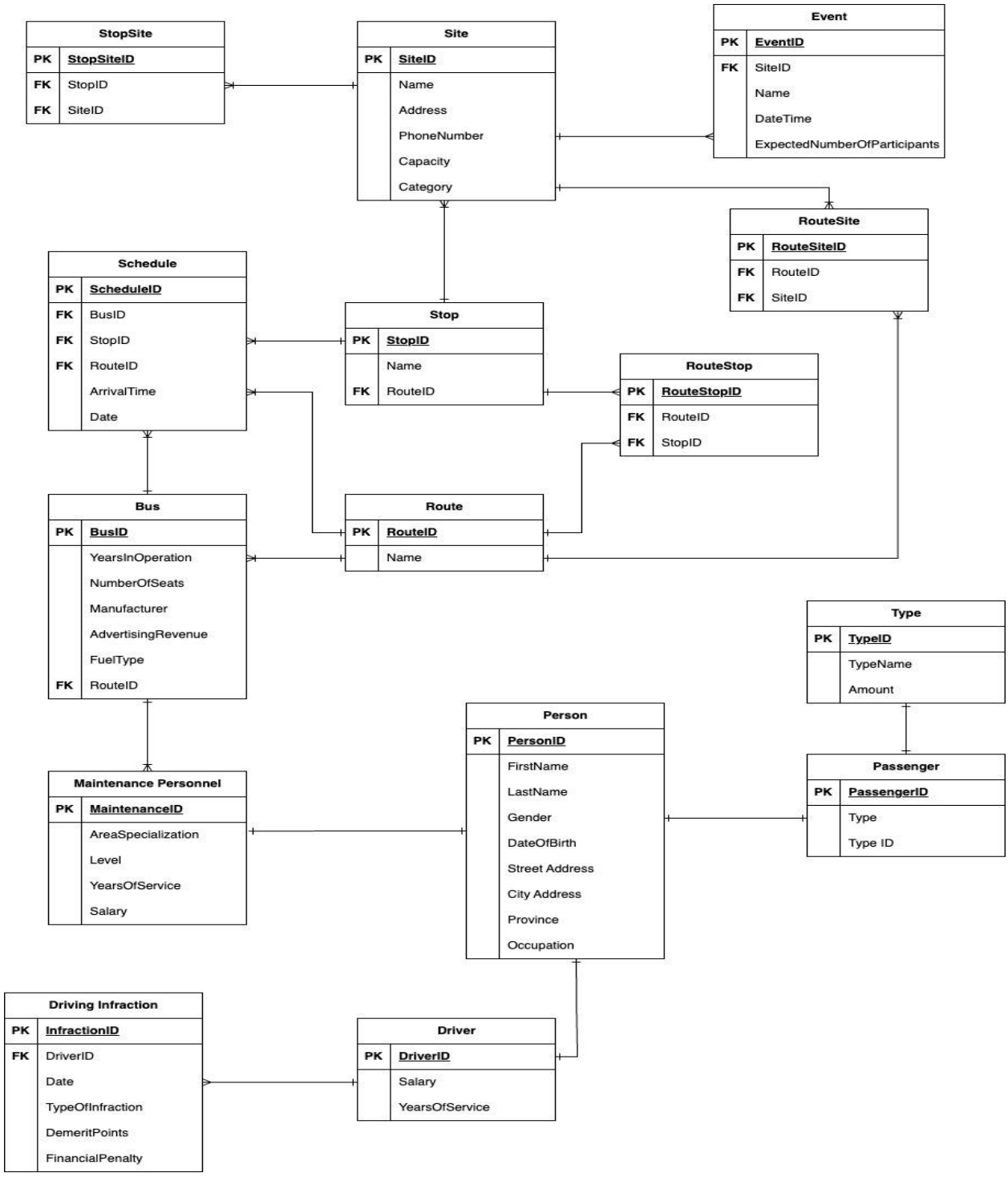
Logic behind Bus and Route-



Logic behind Person -



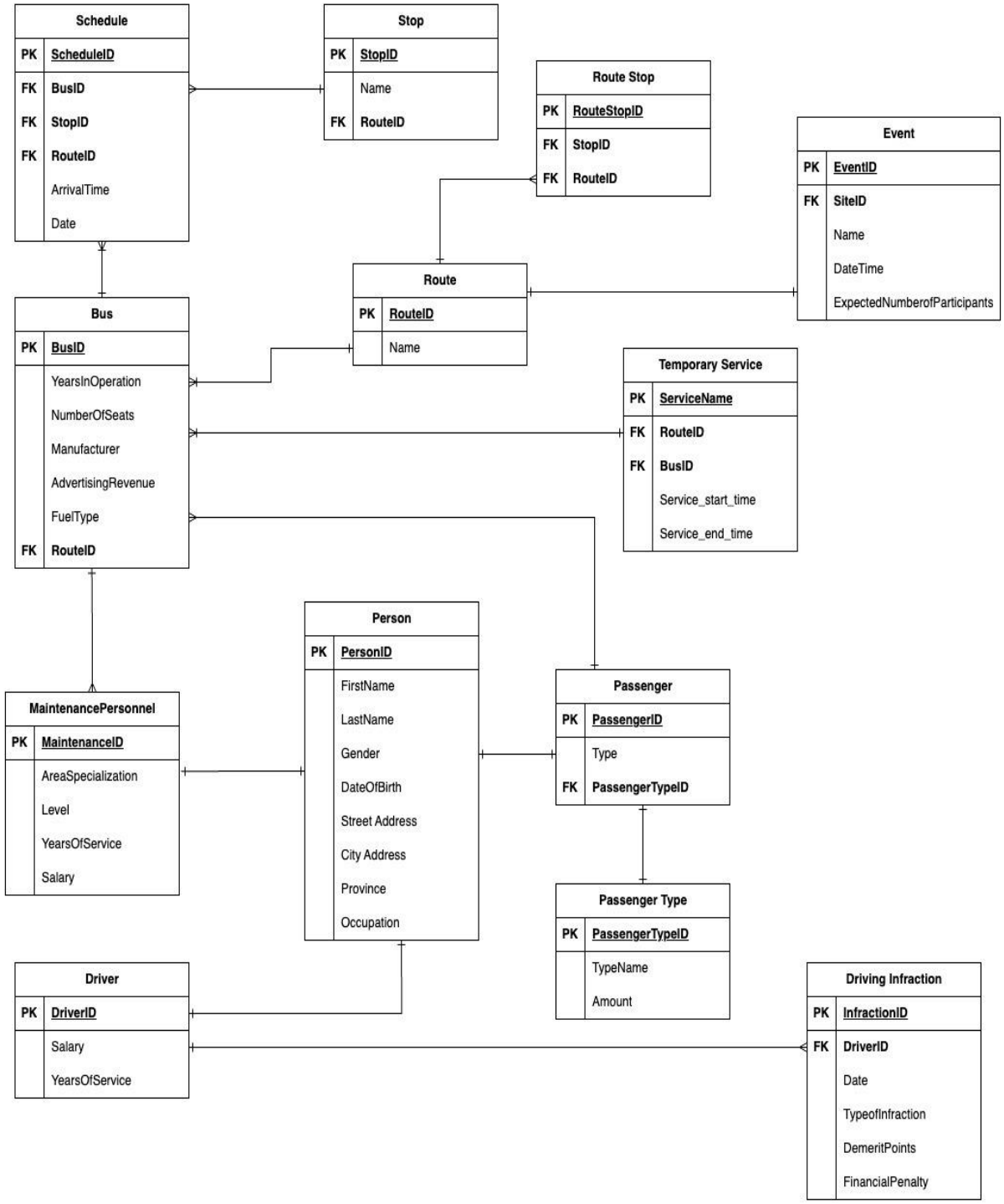
8. Initial Entity Relationship Diagram



9. Finding & Enhancement in Initial ERD

- **Normalization**
  We discovered that the initial ERD has few redundant address data, wherethere can be more than one customer with the same address(households),so we came up with a specific entity to deal with all the addresses of the customers and dealers with an Address# (address id) attribute.
- **Constraints**
  We added a few key constraints, indexes, and sequences to maintain the data integrity.
- **Data Reliability**
  The data entered into the database is made sure that it follows all the rules and we have created a robust database.

10. Final Entity Relationship Diagram LCMDB



11. Business Rule

There are a few basic requirements provided by the business which we considerto be our business rules, which are as follows:

- Event Information: Each event representing a closure or disruption in a specific route must have a unique identifier, name, description, start date and time, end date and time, and affected route(s) recorded.

- Closure Events: Events representing closures or disruptions must specify the affected route(s) and the duration of the closure (start and end date and time).

- Bus Information: Each bus must have a unique identifier, years in operation, number of seats, manufacturer, advertising revenue, and fuel type recorded.

- Alternate Routes: During closures, passengers must be informed about alternate routes or transportation options to minimize inconvenience.

- Coordination with Maintenance: Closure events may be initiated due to maintenance activities, and there must be coordination between maintenance personnel and operations staff to ensure timely completion of maintenance work and restoration of service.

- No Service during Closure: During the duration of a closure event, no buses will operate on the affected route(s).

- Communication of Closure: Closure events must be communicated to passengers through appropriate channels (e.g., website, mobile app, signage at stops) to inform them about service disruptions.

- Sites must have their name, address, phone number, capacity, and category recorded.

12. Script

```
CREATE TABLE Person (
   PersonID NUMBER(20) PRIMARY KEY,
   FirstName VARCHAR2(100),
   LastName VARCHAR2(100),
   Gender VARCHAR2(10),
   DateOfBirth NUMBER(8),
   StreetAddress VARCHAR2(200),
   CityAddress VARCHAR2(100),
   Province VARCHAR2(100),
   Occupation VARCHAR2(100)
);

CREATE TABLE Passenger (
   PassengerID NUMBER(20) PRIMARY KEY,
   PassengerType VARCHAR2(100),
   PassengerTypeID NUMBER(20)
);


CREATE TABLE Driver (
   DriverID NUMBER(20) PRIMARY KEY,
   Salary NUMBER(20),
   YearsOfService NUMBER(20)
);


CREATE TABLE PassengerType (
   PassengerTypeID NUMBER(20) PRIMARY KEY,
   TypeName VARCHAR2(100),
```

```
    Amount NUMBER(10, 2)
);

CREATE TABLE MaintenancePersonnel (
    MaintenanceID NUMBER PRIMARY KEY,
    AreaSpecialization VARCHAR2(100),
    "Level" NUMBER,
    YearsOfService NUMBER,
    Salary NUMBER(20,2)
);

CREATE TABLE Route (
    RouteID NUMBER(20) PRIMARY KEY,
    Name VARCHAR2(100)
);

CREATE TABLE Event (
    EventID NUMBER PRIMARY KEY,
    RouteID NUMBER,
    Name VARCHAR2(100),
    DateTime TIMESTAMP,
    ExpectedNumberOfParticipants NUMBER,
    CONSTRAINT fk_Event_Route FOREIGN KEY (RouteID) REFERENCES
      Route(RouteID)
);

CREATE TABLE DrivingInfraction (
    InfractionID NUMBER(20) PRIMARY KEY,
    DriverID NUMBER(20),
    "Date" DATE,
    TypeOfInfraction VARCHAR2(100),
```

```
    DemeritPoints NUMBER(20),
    FinancialPenalty NUMBER(20),
    FOREIGN KEY (DriverID) REFERENCES Driver(DriverID)
);

CREATE TABLE Bus (
    BusID NUMBER(20) PRIMARY KEY,
    YearsInOperation NUMBER(20),
    NumberOfSeats NUMBER(20),
    Manufacturer VARCHAR2(100),
    AdvertisingRevenue NUMBER(20),
    FuelType VARCHAR2(50),
    RouteID NUMBER(20),
    FOREIGN KEY (Routeid) REFERENCES Route(Routeid)
);


CREATE TABLE Stop (
    StopID NUMBER PRIMARY KEY,
    Name VARCHAR2(100),
    RouteID NUMBER,
    FOREIGN KEY (Routeid) REFERENCES Route(Routeid)
);

CREATE TABLE Schedule (
    ScheduleID NUMBER PRIMARY KEY,
    BusID NUMBER,
    StopID NUMBER,
    RouteID NUMBER,
    ArrivalTime TIMESTAMP,
    "Date" DATE,
```

```
    FOREIGN KEY (Routeid) REFERENCES Route(Routeid),
    FOREIGN KEY (Stopid) REFERENCES Stop(Stopid),
    FOREIGN KEY (Busid) REFERENCES Bus(Busid)
);


CREATE TABLE RouteStop (
    RouteStopID NUMBER PRIMARY KEY,
    RouteID NUMBER,
    StopID NUMBER,
    FOREIGN KEY (Routeid) REFERENCES Route(Routeid),
    FOREIGN KEY (Stopid) REFERENCES Stop(Stopid)
);

CREATE TABLE TempService (
    Service_name VARCHAR2(100) PRIMARY KEY,
    Routeid NUMBER(20),
    Busid NUMBER(20),
    Service_start_time TIMESTAMP,
    Service_end_time TIMESTAMP,
    FOREIGN KEY (Routeid) REFERENCES Route(Routeid),
    FOREIGN KEY (Busid) REFERENCES Bus(Busid)
);
COMMIT;

------------------inser sample records (do not include in final script)--------
INSERT INTO Person (PersonID, FirstName, LastName, Gender, DateOfBirth,
    StreetAddress, CityAddress, Province, Occupation)
VALUES
(1, 'Saahil', 'Karnik', 'Male', 19800101, '123 Keele St', 'Etobicoke', 'Ontario', 'Sr.
    Engineer');
```

```
INSERT INTO Person (PersonID, FirstName, LastName, Gender, DateOfBirth,
    StreetAddress, CityAddress, Province, Occupation)
VALUES(2, 'Mohsin', 'Ahmed', 'Male', 19850215, '456 Queen St', 'North York',
    'Ontario', 'Security');
INSERT INTO Person (PersonID, FirstName, LastName, Gender, DateOfBirth,
    StreetAddress, CityAddress, Province, Occupation)
VALUES(3, 'Zaheeruddin', 'M', 'Male', 19780320, '789 Steeles Ave', 'Markham',
    'Ontario', 'Techie');

INSERT INTO Passenger (PassengerID, PassengerType, PassengerTypeID)
VALUES (1, 'Regular', 1);
INSERT INTO Passenger (PassengerID, PassengerType, PassengerTypeID)
VALUES (2, 'Business', 2);
INSERT INTO Passenger (PassengerID, PassengerType, PassengerTypeID)
VALUES (3, 'Premium', 3);
INSERT INTO Passenger (PassengerID, PassengerType, PassengerTypeID)
VALUES (4, 'Regular', 1);
INSERT INTO Passenger (PassengerID, PassengerType, PassengerTypeID)
VALUES (5, 'Business', 2);

INSERT INTO Driver (DriverID, Salary, YearsOfService)
VALUES
(1, 50000, 3);
INSERT INTO Driver (DriverID, Salary, YearsOfService)
VALUES(2, 60000, 7);
INSERT INTO Driver (DriverID, Salary, YearsOfService)
VALUES(3, 55000, 5);
INSERT INTO Driver (DriverID, Salary, YearsOfService)
VALUES(4, 52000, 4);
INSERT INTO Driver (DriverID, Salary, YearsOfService)
VALUES(5, 58000, 6);
```

```
INSERT INTO PassengerType (PassengerTypeID, TypeName, Amount)
VALUES
(1, 'Regular', 50.00);
INSERT INTO PassengerType (PassengerTypeID, TypeName, Amount)
VALUES(2, 'Business', 100.00);
INSERT INTO PassengerType (PassengerTypeID, TypeName, Amount)
VALUES(3, 'Premium', 150.00);
INSERT INTO PassengerType (PassengerTypeID, TypeName, Amount)
VALUES(4, 'Student', 40.00);
INSERT INTO PassengerType (PassengerTypeID, TypeName, Amount)
VALUES(5, 'Senior', 45.00);

INSERT INTO MaintenancePersonnel (MaintenanceID, AreaSpecialization,
    "Level", YearsOfService, Salary)
VALUES (1, 'Electrical', 2, 4, 60000.00);
INSERT INTO MaintenancePersonnel (MaintenanceID, AreaSpecialization,
    "Level", YearsOfService, Salary)
VALUES (2, 'Mechanical', 3, 6, 70000.00);
INSERT INTO MaintenancePersonnel (MaintenanceID, AreaSpecialization,
    "Level", YearsOfService, Salary)
VALUES (3, 'HVAC', 1, 2, 50000.00);
INSERT INTO MaintenancePersonnel (MaintenanceID, AreaSpecialization,
    "Level", YearsOfService, Salary)
VALUES (4, 'Plumbing', 2, 5, 65000.00);
INSERT INTO MaintenancePersonnel (MaintenanceID, AreaSpecialization,
    "Level", YearsOfService, Salary)
VALUES (5, 'Structural', 3, 7, 75000.00);

INSERT INTO Route (RouteID, Name) VALUES (1, 'Route A');
INSERT INTO Route (RouteID, Name) VALUES (2, 'Route B');
INSERT INTO Route (RouteID, Name) VALUES (3, 'Route C');
```

```
INSERT INTO Route (RouteID, Name)
VALUES (4, 'Route 4');

INSERT INTO Event (EventID, RouteID, Name, DateTime,
    ExpectedNumberOfParticipants)
VALUES (1, 1, 'Event 1', TIMESTAMP '2024-04-10 14:00:00', 50);
INSERT INTO Event (EventID, RouteID, Name, DateTime,
    ExpectedNumberOfParticipants)
VALUES (2, 2, 'Event 2', TIMESTAMP '2024-04-15 18:30:00', 100);
INSERT INTO Event (EventID, RouteID, Name, DateTime,
    ExpectedNumberOfParticipants)
VALUES (3, 3, 'Event 3', TIMESTAMP '2024-04-20 09:00:00', 75);
INSERT INTO Event (EventID, RouteID, Name, DateTime,
    ExpectedNumberOfParticipants)
VALUES (4, 4, 'Event 4', TIMESTAMP '2024-04-11 16:00:00', 50);
INSERT INTO Event (EventID, RouteID, Name, DateTime,
    ExpectedNumberOfParticipants)
VALUES (5, 4, 'Event 5', TIMESTAMP '2024-04-11 16:00:00', 75);
COMMIT;


----------SEQUENCE----------

CREATE SEQUENCE PersonID_Seq
START WITH 1
INCREMENT BY 1;


INSERT INTO Person (PersonID, FirstName, LastName, Gender, DateOfBirth,
    StreetAddress, CityAddress, Province, Occupation)
VALUES (PersonID_Seq.NEXTVAL, 'Karan', 'Parmar', 'Male', 19990808, '123
    Main St', 'Sherbourne', 'Toronto', 'Engineer');
```

```
INSERT INTO Person (PersonID, FirstName, LastName, Gender, DateOfBirth,
    StreetAddress, CityAddress, Province, Occupation)
VALUES (PersonID_Seq.NEXTVAL, 'Aman', 'Khan', 'Male', 19981226, '456 Elm
    St', 'Eglington', 'Toronto', 'Doctor');

INSERT INTO Person (PersonID, FirstName, LastName, Gender, DateOfBirth,
    StreetAddress, CityAddress, Province, Occupation)
VALUES (PersonID_Seq.NEXTVAL, 'Shrajna', 'Shetty', 'Female', 19980815, '125
    Village Green Sq', 'Scarborough', 'Toronto', 'Engineer');
COMMIT;

--------------------VIEW--------------------
---Simple View----

CREATE VIEW PersonView AS
SELECT PersonID, FirstName, LastName, Gender, Occupation
FROM Person;
COMMIT;

CREATE VIEW PassengerInfo AS
SELECT PassengerID, PassengerType, PassengerTypeID
FROM Passenger;
COMMIT;

-----Complex View-----
CREATE VIEW EventDetails AS
SELECT e.EventID, e.Name AS EventName, r.Name AS RouteName,
    e.DateTime, e.ExpectedNumberOfParticipants
FROM Event e
INNER JOIN Route r ON e.RouteID = r.RouteID;
COMMIT;
```

```
--------------------CURSOR/EXCEPTION--------------------
CREATE OR REPLACE PROCEDURE InsertOrUpdateDriver(
   p_DriverID IN NUMBER,
   p_Salary IN NUMBER,
   p_YearsOfService IN NUMBER
) AS
   v_SalaryCheck NUMBER;
   v_ErrorMsg VARCHAR2(200);
BEGIN
   -- Check if the salary is negative
   IF p_Salary < 0 THEN
      v_ErrorMsg := 'Salary cannot be negative.';
      RAISE_APPLICATION_ERROR(-20001, v_ErrorMsg);
   END IF;

   -- Check if the driver already exists
   SELECT COUNT(*) INTO v_SalaryCheck FROM Driver WHERE DriverID =
    p_DriverID;

   IF v_SalaryCheck = 0 THEN
      -- Insert new driver
      INSERT INTO Driver (DriverID, Salary, YearsOfService) VALUES
    (p_DriverID, p_Salary, p_YearsOfService);
   ELSE
      -- Update existing driver
      UPDATE Driver SET Salary = p_Salary, YearsOfService = p_YearsOfService
    WHERE DriverID = p_DriverID;
   END IF;
EXCEPTION
   WHEN DUP_VAL_ON_INDEX THEN
      v_ErrorMsg := 'Driver with ID ' || p_DriverID || ' already exists.';
```

```
        RAISE_APPLICATION_ERROR(-20002, v_ErrorMsg);
    WHEN OTHERS THEN
        v_ErrorMsg := 'An error occurred: ' || SQLERRM;
        RAISE_APPLICATION_ERROR(-20003, v_ErrorMsg);
END;
COMMIT;

----calling procedure----

BEGIN
    InsertOrUpdateDriver(1, 9000, 5);  -- Insert new driver with positive salary
    InsertOrUpdateDriver(2, -1000, 3);  -- Try to insert with negative salary (will
     raise exception)
EXCEPTION
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('Error: ' || SQLERRM);
END;
COMMIT;

--------------------FUNCTIONS---------------------
CREATE OR REPLACE FUNCTION CalculateFare(
    p_passenger_type_id IN NUMBER,
    p_distance_travelled IN NUMBER
)
RETURN NUMBER
IS
    v_amount NUMBER(10, 2);
BEGIN
    SELECT Amount INTO v_amount
    FROM PassengerType
    WHERE PassengerTypeID = p_passenger_type_id;
```

```
    RETURN v_amount * p_distance_travelled;
END CalculateFare;
COMMIT;

-- Function to Calculate Age
CREATE OR REPLACE FUNCTION CalculateAge(dateOfBirth NUMBER)
RETURN NUMBER
IS
    age NUMBER;
BEGIN
    SELECT TRUNC(MONTHS_BETWEEN(SYSDATE, TO_DATE(dateOfBirth,
      'YYYYMMDD')) / 12)
    INTO age
    FROM dual;

    RETURN age;
END;
COMMIT;

CREATE OR REPLACE PROCEDURE AssignPassengerType(personID NUMBER)
IS
    passengerTypeID NUMBER;
    age NUMBER;
BEGIN
    SELECT CalculateAge(DateOfBirth) INTO age FROM Person WHERE PersonID
      = personID;
    IF age < 18 THEN
        SELECT PassengerTypeID INTO passengerTypeID FROM PassengerType
      WHERE TypeName = 'Child';
    ELSIF age >= 18 AND age < 65 THEN
```

```
      SELECT PassengerTypeID INTO passengerTypeID FROM PassengerType
     WHERE TypeName = 'Adult';
   ELSE
      SELECT PassengerTypeID INTO passengerTypeID FROM PassengerType
     WHERE TypeName = 'Senior Citizen';
   END IF;
   UPDATE Passenger SET PassengerTypeID = passengerTypeID WHERE
     PassengerID = personID;

   DBMS_OUTPUT.PUT_LINE('Passenger Type Assigned Successfully.');
EXCEPTION
   WHEN NO_DATA_FOUND THEN
      DBMS_OUTPUT.PUT_LINE('Error: Passenger Type not found.');
   WHEN OTHERS THEN
      DBMS_OUTPUT.PUT_LINE('Error: An unexpected error occurred.');
END;
COMMIT;

--------------------TRIGGERS---------------------

CREATE OR REPLACE TRIGGER trg_Passenger_TypeCheck
BEFORE INSERT OR UPDATE OF PassengerTypeID ON Passenger
FOR EACH ROW
DECLARE
   v_TypeID PassengerType.PassengerTypeID%TYPE;
BEGIN
   SELECT PassengerTypeID INTO v_TypeID
   FROM PassengerType
   WHERE PassengerTypeID = :NEW.PassengerTypeID;

   IF v_TypeID IS NULL THEN
```

```
      RAISE_APPLICATION_ERROR(-20001, 'Invalid PassengerTypeID. Please
    provide a valid PassengerTypeID.');
  END IF;
END;
COMMIT;

CREATE OR REPLACE TRIGGER trg_Bus_RouteCheck
BEFORE INSERT OR UPDATE OF RouteID ON Bus
FOR EACH ROW
DECLARE
  v_RouteID Route.RouteID%TYPE;
BEGIN
  SELECT RouteID INTO v_RouteID
  FROM Route
  WHERE RouteID = :NEW.RouteID;

  IF v_RouteID IS NULL THEN
    RAISE_APPLICATION_ERROR(-20002, 'Invalid RouteID. Please provide a
    valid RouteID.');
  END IF;
END;
COMMIT;

CREATE OR REPLACE TRIGGER trg_Schedule_IDCheck
BEFORE INSERT OR UPDATE OF BusID, StopID, RouteID ON Schedule
FOR EACH ROW
DECLARE
  v_BusID Bus.BusID%TYPE;
  v_StopID Stop.StopID%TYPE;
  v_RouteID Route.RouteID%TYPE;
BEGIN
```

```
    SELECT BusID INTO v_BusID
    FROM Bus
    WHERE BusID = :NEW.BusID;

    SELECT StopID INTO v_StopID
    FROM Stop
    WHERE StopID = :NEW.StopID;

    SELECT RouteID INTO v_RouteID
    FROM Route
    WHERE RouteID = :NEW.RouteID;

    IF v_BusID IS NULL OR v_StopID IS NULL OR v_RouteID IS NULL THEN
       RAISE_APPLICATION_ERROR(-20003, 'Invalid BusID, StopID, or RouteID.
     Please provide valid IDs.');
    END IF;
END;
COMMIT;

--------------------TRIGGERS---------------------

CREATE OR REPLACE TRIGGER trg_Passenger_TypeCheck
BEFORE INSERT OR UPDATE OF PassengerTypeID ON Passenger
FOR EACH ROW
DECLARE
    v_TypeID PassengerType.PassengerTypeID%TYPE;
BEGIN
    SELECT PassengerTypeID INTO v_TypeID
    FROM PassengerType
    WHERE PassengerTypeID = :NEW.PassengerTypeID;
```

```
    IF v_TypeID IS NULL THEN
        RAISE_APPLICATION_ERROR(-20001, 'Invalid PassengerTypeID. Please
      provide a valid PassengerTypeID.');
    END IF;
END;
COMMIT;

CREATE OR REPLACE TRIGGER trg_Bus_RouteCheck
BEFORE INSERT OR UPDATE OF RouteID ON Bus
FOR EACH ROW
DECLARE
    v_RouteID Route.RouteID%TYPE;
BEGIN
    SELECT RouteID INTO v_RouteID
    FROM Route
    WHERE RouteID = :NEW.RouteID;

    IF v_RouteID IS NULL THEN
        RAISE_APPLICATION_ERROR(-20002, 'Invalid RouteID. Please provide a
      valid RouteID.');
    END IF;
END;
COMMIT;

CREATE OR REPLACE TRIGGER trg_Schedule_IDCheck
BEFORE INSERT OR UPDATE OF BusID, StopID, RouteID ON Schedule
FOR EACH ROW
DECLARE
    v_BusID Bus.BusID%TYPE;
    v_StopID Stop.StopID%TYPE;
    v_RouteID Route.RouteID%TYPE;
```

```
BEGIN
   SELECT BusID INTO v_BusID
   FROM Bus
   WHERE BusID = :NEW.BusID;

   SELECT StopID INTO v_StopID
   FROM Stop
   WHERE StopID = :NEW.StopID;

   SELECT RouteID INTO v_RouteID
   FROM Route
   WHERE RouteID = :NEW.RouteID;

   IF v_BusID IS NULL OR v_StopID IS NULL OR v_RouteID IS NULL THEN
      RAISE_APPLICATION_ERROR(-20003, 'Invalid BusID, StopID, or RouteID.
     Please provide valid IDs.');
   END IF;
END;
COMMIT;


--------------------PACKAGES---------------------

CREATE OR REPLACE PACKAGE PassengerPackage AS
   TYPE PassengerRecType IS RECORD (
      PassengerID NUMBER(20),
      FirstName VARCHAR2(100),
      LastName VARCHAR2(100),
      Gender VARCHAR2(10),
      DateOfBirth NUMBER(8),
      StreetAddress VARCHAR2(200),
```

```
    CityAddress VARCHAR2(100),
    Province VARCHAR2(100),
    Occupation VARCHAR2(100)
);

PROCEDURE AddPassenger(
    p_PassengerData IN PassengerRecType
);

PROCEDURE UpdatePassenger(
    p_PassengerID IN NUMBER,
    p_PassengerData IN PassengerRecType
);

PROCEDURE DeletePassenger(
    p_PassengerID IN NUMBER
);

FUNCTION GetPassengerByID(
    p_PassengerID IN NUMBER
) RETURN PassengerRecType;

-- Exception declarations
DuplicatePassengerException EXCEPTION;
PassengerNotFoundException EXCEPTION;
PRAGMA EXCEPTION_INIT(DuplicatePassengerException, -20001);
PRAGMA EXCEPTION_INIT(PassengerNotFoundException, -20002);
END PassengerPackage;
COMMIT;
```

```
CREATE OR REPLACE PACKAGE DriverPackage AS
   TYPE DriverRecType IS RECORD (
      DriverID NUMBER(20),
      Salary NUMBER(20),
      YearsOfService NUMBER(20)
   );

   PROCEDURE AddDriver(
      p_DriverData IN DriverRecType
   );

   PROCEDURE UpdateDriver(
      p_DriverID IN NUMBER,
      p_DriverData IN DriverRecType
   );

   PROCEDURE DeleteDriver(
      p_DriverID IN NUMBER
   );

   FUNCTION GetDriverByID(
      p_DriverID IN NUMBER
   ) RETURN DriverRecType;

   -- Exception declarations
   DuplicateDriverException EXCEPTION;
   DriverNotFoundException EXCEPTION;
   PRAGMA EXCEPTION_INIT(DuplicateDriverException, -20003);
   PRAGMA EXCEPTION_INIT(DriverNotFoundException, -20004);
END DriverPackage;
COMMIT;
```

```
CREATE OR REPLACE PACKAGE MaintenancePackage AS
  TYPE MaintenanceRecType IS RECORD (
    MaintenanceID NUMBER,
    AreaSpecialization VARCHAR2(100),
    Level NUMBER,
    YearsOfService NUMBER,
    Salary NUMBER(20, 2)
  );

  PROCEDURE AddMaintenancePersonnel(
    p_MaintenanceData IN MaintenanceRecType
  );

  PROCEDURE UpdateMaintenancePersonnel(
    p_MaintenanceID IN NUMBER,
    p_MaintenanceData IN MaintenanceRecType
  );

  PROCEDURE DeleteMaintenancePersonnel(
    p_MaintenanceID IN NUMBER
  );

  FUNCTION GetMaintenancePersonnelByID(
    p_MaintenanceID IN NUMBER
  ) RETURN MaintenanceRecType;

  -- Exception declarations
  DuplicateMaintenanceException EXCEPTION;
  MaintenanceNotFoundException EXCEPTION;
  PRAGMA EXCEPTION_INIT(DuplicateMaintenanceException, -20005);
```

```
    PRAGMA EXCEPTION_INIT(MaintenanceNotFoundException, -20006);
END MaintenancePackage;
COMMIT;

--------------------ANALYTICAL SKILL DEMONSTRATION--------------------


---Query to Calculate Total Salary Expense for Maintenance Personnel---
SELECT SUM(Salary) AS TotalSalaryExpense
FROM MaintenancePersonnel;

---Query to Identify Routes with Low Advertising Revenue---
SELECT RouteID, Name
FROM Route
WHERE RouteID NOT IN (SELECT RouteID FROM Bus WHERE
    AdvertisingRevenue > 1000);

---Query to Calculate Average Salary for Drivers with Over 5 Years of Service---
SELECT AVG(Salary) AS AverageSalary
FROM Driver
WHERE YearsOfService > 5;

---Query to List Events Scheduled for a Specific Route---
SELECT EventID, Name, DateTime
FROM Event
WHERE RouteID = (SELECT RouteID FROM Route WHERE Name = 'Route 4');
```

13. Challenges

The following are the primary challenges faced during development of the database system -

- **Data migration:** The database system by itself is a complex multi-faceted model which resulted in numerous complications during data ingestion since not all the involved systems had the relevant required data. This would be fixed with an efficient integration layer for data ingestion.
- **Scalability:** The volume of the data involved would be huge since the dataranges from potential routes to former passengers. However, with stringent data normalization techniques, the system would be able to handle the complicated data.
- **Data Archival:** an efficient data archival system is needed to store periodicsnapshots of the system to avoid any data loss.

14. Conclusion

Comprehensive business intelligence and the following innovations are prevented from forming without insightful data from the past. The Hamilton Master Database system will be the first and most formidable step for the organization to accurately chart out new and profitable routes while optimizing the resources available, all while keeping in mind the behavior and preferences of its passengers. This will provide the intelligent data needed for the stakeholders with its robust and efficient design.