PROJECT REPORT

DAMG 6210 – DATABASE MANAGEMENT AND DESIGN

Spring Semester 2024

INSTRUCTOR: SYED FARHAN MAZHAR

PROJECT TITLE: TRANSITTRACK

STREAMLINING TRANSIT OPERATIONS THROUGH DATA MANAGEMENT

GROUP THREE

ABDUL AZEEM SYED - 002838227

DANIEL IKECHUKWU - 002617047

HABEEBUDDIN MIR – 00271392

TABLE OF CONTENTS

1. EXECUTIVE SUMMARY

The Transittrack Database Management project aims to streamline transit operations by leveraging the features of a comprehensive data management system concepts learned through the “DAMG:6210 - Database Management and Design” course. We aim to implement all the crucial course outcomes while following the best industry practices to succeed in this final project.

Inspired by the wide and interesting case study of Canada's renowned GO Train System, Transittrack seeks to establish a centralized database that efficiently manages critical transit information. Drawing inspiration from the GO Train System's effective scheduling, passenger services, and maintenance procedures, our project endeavors to optimize transit operations, enhance passenger experiences, and ensure system reliability. By adopting best practices from the GO Train System and leveraging modern technology, Transittrack aspires to set new standards in transit management and efficiency.

The implementation began with a thorough analysis of requirements and insights from transit authorities, stakeholders, and domain experts. The database design phase focusses on a robust schema that ensures data integrity, scalability, and security. During implementation, database tables, indices, and procedures are researched and developed, followed by rigorous testing to validate functionality and performance. Multiple versions of Entity Relationship Diagrams were made, and considerable time and efforts were dedicated into integrating all the submission feedback received throughout the semester.

2. INTRODUCTION

2.1 OUTLINE

The project report follows the standards laid out in the coursework and as stated by the professor during sessions. Commencing with Requirement Gathering, which emphasizes the significance of understanding stakeholders' needs and operational requirements. This phase delves into the methodologies employed for gathering inputs, defining Business Rules that serve as guidelines for data management, outlining Key Deliverables expected at the end of this phase, and discussion with regards to the Tools and Techniques utilized to implement the project.

Moving forward, the Design and Modeling section navigates through the details of database architecture. Conceptual Modeling elaborates the high-level representation of the database schema, Logical Modeling translates this abstraction into a more structured form, and Physical Database Design maps the actual implementation details, considering factors like storage optimization and indexing strategies.

The subsequent section navigates the Entity Relationship Diagram (ERD), projecting its evolution from the Initial ERD to the Final ERD post-analysis and normalization. Major Design Elements further dive deeply into the core components of the database, from Entities, to Tables, detailing their attributes and relationships, and finally Relationships/Associations, emphasizing the connections forged between various entities.

Database Scripts highlights the technical aspects of implementation, providing scripts for Database Definition Language (DDL) to define schema elements, Data Manipulation Language (DML) for data manipulation, Views for simplified data access, and Common Queries for retrieving information efficiently.

Challenges encountered during the project, such as requirement translation complexities or performance optimization hurdles, are candidly discussed. The Conclusion section summarizes the project's outcomes and implications for transit operations. References and an Appendix containing supplementary material provide further depth and context to the report.

2.2 REQUIREMENT GATHERING

The data requirements and gathering phase for implementing Canada's GO Train System Database involves a detailed process of identifying, analyzing, and documenting the data needs of the project to ensure that the database effectively supports the operational, analytical, and decision-making requirements of Metrolinx and its stakeholders. This phase typically includes the following key steps:

2.2.1 Stakeholder Engagement: The project team engages with various stakeholders, including Metrolinx management, operational staff, IT professionals, and external partners, to understand their data needs, priorities, and concerns. Stakeholder input is crucial for identifying critical data elements and ensuring alignment with organizational goals and objectives.

2.2.2 Data Collection and Analysis: Comprehensive data collection is conducted to gather information about the existing data sources, formats, quality, and governance practices within Metrolinx. This includes reviewing documentation, conducting interviews, and analyzing sample datasets to identify relevant data entities, attributes, relationships, and constraints.

2.2.3 Functional Requirements Gathering: The project team collaborates with stakeholders to elicit functional requirements that specify the data manipulation, retrieval, reporting, and analysis capabilities required from the database. This involves defining use cases, scenarios, and user stories to capture the desired functionality and user interactions with the system.

2.2.4 Non-Functional Requirements Definition: In addition to functional requirements, non-functional requirements are identified to specify quality attributes such as performance, scalability, security, reliability, and compliance with regulatory standards. These requirements guide the design and implementation of the database architecture, data models, and access controls.
The requirements gathered through the implementation of the above methods are presented in the table below.

Requirement Type	Requirement
Functional	● Ability to create, update, and delete records for all entities. ● Capability to search, filter, and retrieve data efficiently. ● Feature to perform functions based on stored data ● Ability to maintain data consistency and integrity across the database. ● Each entity must have a unique identifier (e.g., train number, station code). ● Data attributes must adhere to specified constraints (e.g., capacity > 0, fare >= 0). ● Status or type attributes must be predefined and categorized (e.g., operational, out of service). ● Relationships between entities must be well-defined and valid (e.g., ticket associated with valid passenger).
Non-Functional	● Performance: Ensure efficient query processing and response times for data retrieval. ● Reliability: Maintain high availability and data integrity to minimize downtime and errors. ● Security: Implement encryption mechanisms to protect sensitive data, with access controls and user authentication ● Scalability: Design the database to handle increasing data volume and user traffic, with efficient indexing and partitioning strategies.

2.3 BUSINESS RULES

The requirements gathered in the previous sections provide us with the layout to define the business rules. Business rules serve as fundamental guidelines that govern the behavior, operations, and data management within the Transittrack database. These rules are essential for ensuring data consistency, integrity, and accuracy, and they help enforce standardized practices across the transit system. Business rules are defined based on the specific requirements and objectives of the stakeholders and transit authorities, reflecting their operational policies and regulatory requirements. The Business Rules defined for our projected are listed as follows:

2.3.1 Train Entity:

- *Each train must have a unique train number and should not be null.*
- *The capacity of a train must be greater than zero.*
- *The status of a train can be either "operational" or "out of service".*
- *Each train should be associated with a route.*

2.3.2 Station Entity:

- *Each station must have a unique station code and should not be null.*
- *Station address (street address, city) should not be null.*
- *Station name should not be null.*

2.3.3 Route Entity:

- *Each route must have a unique route ID.*
- *A route must consist of two distinct stations (start and end stations).*
- *The distance between start and end stations must be greater than zero.*

2.3.4 Schedule Entity:

- *Each schedule must have a unique schedule ID.*
- *Schedules should be assigned to a valid route.*
- *Departure time should not be null.*

2.3.5 Passenger Entity:

- *Each passenger must have a unique passenger ID.*
- *Passenger names should not be null and should adhere to specified character limits.*
- *Contact information for passengers should include valid email addresses.*
- *Each passenger should be associated with a valid fare type.*

2.3.6 Ticket Entity:

- *Each ticket must have a unique ticket ID.*
- *Tickets should be associated with the valid route.*
- *People outside of the system can also buy a ticket. So, passenger value can be null.*
- *The fare for a ticket should be greater than or equal to zero.*
- *The default value for purchase time should be current time.*

2.3.7 Employee Entity:

- *Each employee must have a unique employee ID.*
- *Employee names should not be null and should adhere to specified character limits.*
- *Contact information for employees should include valid email addresses.*
- *Employee should be associated with valid employee type.*

2.3.8 Maintenance Entity:

- *Each maintenance activity must have a unique maintenance ID.*
- *Maintenance activities should be associated with trains.*
- *The last maintenance date should not be null.*

2.3.9 Incident Entity:

- *Each incident must have a unique incident ID.*
- *Incident description should not be null.*
- *Incidents should be associated with valid alerts.*

2.3.10 Fare Type Entity:

- *Each fare type must have a unique fare ID.*
- *Discounts offered for fare types should be clearly defined and more than or equal to 0.*

2.3.11 Accessibility Feature Entity:

- *Each accessibility feature must have a unique feature ID.*
- *Feature type should not be null.*
- *Flag for feature availablity should not be null.*

2.3.12 Alert Entity:

- *Each alert must have a unique alert ID.*
- *Alert Type should be predefined and categorized (e.g., Delay, Cancelation)*
- *Each alert should be associated with a schedule.*

2.3.13 Performance Metrics Entity:

- *Each performance metric must have a unique metric ID.*
- *Description and Value should not be null.*
- *Metrics should be associated with a valid station.*

2.3.14 Customer Support Entity:

- *Each customer support must have a unique support ID.*
- *Subject, Status and description cannot be null.*
- *Each support ticket should be associated with a valid Passenger and Employee.*

2.4 KEY DELIVERABLES

2.4.1 Database Schema: The database schema serves as the blueprint for the structure of the Transit System Database. It includes the definition of tables, attributes, relationships, constraints, and indexes based on the identified entities, attributes, and relationships from the requirements and business rules.

2.4.2 Entity Relationship Diagram (ERD): The ERD visually represents the relationships between entities in the Transit System Database. It provides a clear overview of how different entities are connected and organized within the database, helping stakeholders understand the data model and its implications for transit operations.

2.4.3 Data Dictionary: The data dictionary documents the definitions and characteristics of data elements in the Transit System Database. It provides detailed descriptions of each entity, attribute, data type, constraint, and business rule, serving as a reference guide for database administrators, developers, and users.

2.4.4 Business Rules: The functional requirements specification outlines the specific features and capabilities of the Transit System Database. It documents the functional requirements derived from the business rules and requirements gathering phase, detailing the expected behavior and functionality of the database solution.

2.4.5 Conceptual Modeling: Conceptual modeling involves creating high-level representations of the Transit System Database, focusing on the entities, attributes, and relationships without delving into technical implementation details. It provides stakeholders with a conceptual understanding of the database structure and functionality.

2.4.6 Logical Modeling: Logical modeling translates the conceptual model into a more structured and detailed representation, defining entity relationships, data attributes, and normalization to ensure data integrity and efficiency. It serves as the basis for the physical database design.

2.4.7 Physical Database Design: Physical database design involves implementing the logical model into a physical database schema, considering factors such as data storage, indexing, partitioning, and performance optimization. It defines the actual database objects and configurations required for the Transit System Database.

2.4.8 Implementation Plan: The implementation plan outlines the steps and timeline for deploying the Transit System Database into production. It includes tasks, dependencies, resources, and milestones for database installation, configuration, data migration, testing, training, and rollout, ensuring a smooth and successful deployment process.

2.5 TOOLS AND TECHNIQUES

2.5.1 Oracle SQL Developer: An integrated development environment (IDE) tailored for Oracle databases, facilitating database design, SQL coding, debugging, and performance tuning.

2.5.2 Oracle Database Express Edition (XE): A free, lightweight version of Oracle Database, suitable for development and small-scale production deployments, providing robust features for data modeling, SQL querying, and database administration.

2.5.3 Oracle SQL Developer Data Modeler: A powerful tool for designing and documenting database schemas, enabling the creation and visualization of entity-relationship diagrams (ERDs), and generation of SQL scripts for database creation.

2.5.4 Entity Relationship Diagram (ERD) Tool: For this we chose draw[dot]io. Draw.io is great for ERDs because it's easy to use with drag-and-drop features, offers extensive customization options, supports real-time collaboration, integrates with popular platforms, is accessible from any device, and provides version control for tracking changes.

2.5.5 Normalization Techniques: Functional dependency analysis, normalization forms (e.g., 1NF, 2NF, 3NF), and denormalization strategies are employed during the logical modeling phase to ensure data integrity, minimize redundancy, and optimize database performance.

2.5.6 Requirement Gathering Techniques: Techniques such as interviews, surveys, workshops, and documentation review are employed to gather functional and non-functional requirements for the Transit System Database, involving stakeholders to understand their needs, preferences, and constraints.

2.5.7 Team Collaboration and Version Controlling: Utilizing collaboration platform GitHub facilitates team collaboration by allowing multiple developers to work on the database project simultaneously. Microsoft Teams and Office are used for team communication and documentation.

2.6 IMPLEMENTATION TIMELINE

Phase	Goals	Timeframe (Days)
Research and Planning	● Literature review and research ● Requirement gathering ● Defining Business Rules ● Finalizing Key Deliverables ● Deciding Tools and Techniques	Eight (8)
Design	● Conceptual Modelling ● Logical Modelling ● Physical Database Design	Eleven (11)
ERD	● Initial ERD ● Analysis and Normalization ● Final ERD	Seven (7)
Review	● Peer and Expert Reviews ● Stakeholder Reviews ● Incorporations of feedback	Four (4)

Scripting	● DDL Scripts ● DML Scripts ● Views ● Sample Test Queries	Eight (8)
Documentation	● Project Report ● Project Presentation	Four (4)

3. DESIGN AND MODELING

3.1 CONCEPTUAL MODELING

During the conceptual modeling phase of the project, the focus is on developing a high-level representation of the database structure and content that aligns with the business requirements and data needs identified in the previous phases. This phase typically involves the following key activities:

3.1.1. Identifying Entities and Relationships: The project team identifies the main entities or objects of interest within the GO Train system, such as trains, stations, schedules, routes, passengers, and maintenance activities. Relationships between these entities are established to depict how they interact with each other. Entity-Relationship Diagrams (ERDs) are used to visually represent these entities and relationships, providing a clear understanding of the database structure.

3.1.2. Defining Attributes and Constraints: For each entity identified, the attributes or properties that describe its characteristics are defined. These attributes include fields such as train number, station name, departure time, passenger name, etc. Additionally, constraints such as uniqueness, mandatory fields, data types, and allowable values are specified to ensure data integrity and consistency.

3.1.3. Modeling Associations and Cardinality: Associations between entities are modeled to represent the nature and degree of the relationships. Cardinality constraints, such as one-to-one, one-to-many, or many-to-many, are defined to specify how instances of one entity relate to instances of another entity. This helps in understanding the navigation paths and data dependencies within the database.

3.2 LOGICAL MODELING

Entity: Train
Attributes:

- TrainID (Primary Key)
- TrainNumber
- Capacity
- Status
- RouteID

Entity: Route
Attributes:

- RouteID (Primary Key)
- StartStationID
- EndStationID
- Distance

Entity: Station
Attributes:

- StationID (Primary Key)
- StationCode
- StationName
- StreetAddress
- City

Entity: Schedule
Attributes:

- ScheduleID (Primary Key)
- DepartureTime
- RouteID

Entity: Passenger
Attributes:

- PassengerID (Primary Key)
- PassengerName
- Email
- FareTypeID

Entity: Ticket
Attributes:

- TicketID (Primary Key)

- RouteID
- PassengerID
- Fare
- PurchaseDate

Entity: Employee
Attributes:

- EmployeeID (Primary Key)
- EmployeeName
- Email
- TypeID

Entity: Maintenance
Attributes:

- MaintenanceID (Primary Key)
- TrainID
- LastMaintenanceDate

Entity: Incident
Attributes:

- IncidentID (Primary Key)
- Description
- AlertID

Entity: Alert
Attributes:

- AlertID (Primary Key)
- AlertType
- Description
- ScheduleID
- TimeAdded

Entity: PerformanceMetrics
Attributes:

- MetricID (Primary Key)
- MetricType
- Value
- StationID

Entity: FareType
Attributes:

- FareTypeID (Primary Key)
- Type
- DiscountPercentage

Entity: AccessibilityFeature
Attributes:

- FeatureID (Primary Key)
- Type
- IsAvailable
- StationID
-

Entity: CustomerSupport
Attributes:

- SupportID (Primary Key)
- Status
- Subject
- Description
- PassengerID
- EmployeeID

3.3 PHYSICAL DATABASE DESIGN

This physical database design maps the logical entities and attributes to actual database tables, specifying data types, primary and foreign keys, and constraints. It considers implementation details relevant to the Oracle SQL database platform, ensuring efficient storage and retrieval of data for the Transit System Database.

Table Name	Columns
Train	- TrainID (Primary Key, NUMBER) - TrainNumber (VARCHAR) - Capacity (NUMBER) - Status (VARCHAR) - RouteID (Foreign Key, NUMBER)

Route	- RouteID (Primary Key, NUMBER) - Name (VARCHAR) - StartStationID (Foreign Key, NUMBER) - EndStationID (Foreign Key, NUMBER) - Distance (FLOAT)
Station	- StationID (Primary Key, NUMBER) - StationCode (VARCHAR) - Name (VARCHAR) - StreetAddress (VARCHAR) - City (VARCHAR)
Schedule	- ScheduleID (Primary Key, NUMBER) - RouteID (Foreign Key, INT) - DepartureTime (TIMESTAMP)
Passenger	- PassengerID (Primary Key, NUMBER) - Name (VARCHAR) - ContactInfo (VARCHAR) - FareTypeID (Foreign Key, NUMBER)
Ticket	- TicketID (Primary Key, NUMBER) - PassengerID (Foreign Key, NUMBER) - RouteID(Foreign Key, NUMBER) - Fare (DECIMAL) - PurchaseTime (TIMESTAMP)
Employee	- EmployeeID (Primary Key, NUMBER) - Name (VARCHAR) - EmployeeTypeID(NUMBER) - ContactInfo (VARCHAR)
EmployeeType	- EmployeeTypeID(Primary Key, NUMBER)

	- Type (VARCHAR)
EmployeeDesignation	- EmployeeDesignationID (Primary Key, NUMBER) - EmployeeID(NUMBER) - TrainID(Foreign Key, NUMBER) - StationID(Foreign Key, NUMBER) - IsCustomerSupport(NUMBER)
Maintenance	- MaintenanceID (Primary Key, NUMBER) - TrainID (NUMBER) -Last MaintenanceDate (DATE)
Incident	- IncidentID (Primary Key, NUMBER) - Description (VARCHAR) - AlertID(Foreign Key, NUMBER)
Alert	- AlertID (Primary Key, NUMBER) - Type (VARCHAR) - Description (VARCHAR) - ScheduleID (Foreign Key, NUMBER) - TimeAdded(NUMBER)
PerformanceMetrics	- MetricID (Primary Key, NUMBER) - Description (VARCHAR) - Value (DECIMAL) - StationID (Foreign Key, NUMBER)
FareType	- FareTypeID (Primary Key, NUMBER) - Name (VARCHAR) - DiscountPercentage(NUMBER)
AccessibilityFeature	- FeatureID (Primary Key, NUMBER) - Description (VARCHAR) - IsAvailable (Boolean)

CustomerSupport	- CustomerSupportID(Primary Key, NUMBER) - Subject (VARCHAR) - Description (VARCHAR) - EmployeeID (Foreign Key, NUMBER) - PassengerID (Foreign Key, NUMBER)

4. ENTITY RELATIONSHIP DIAGRAM

4.1 INITIAL ERD

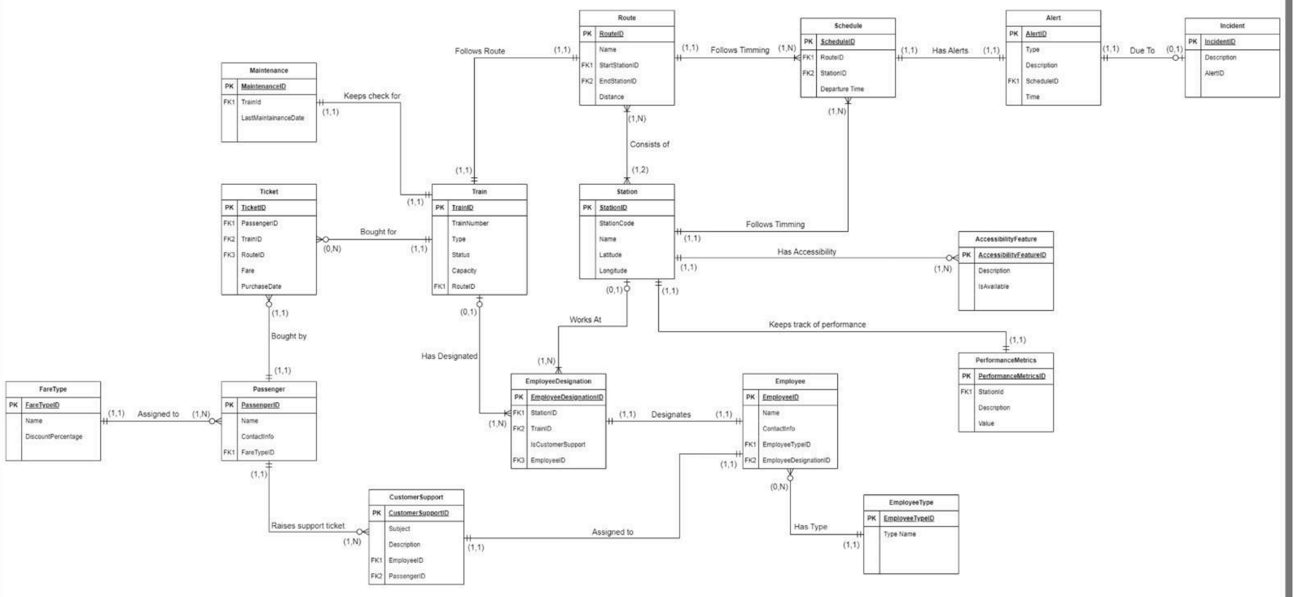


DIAGRAM 4.1. INITIAL ENTITY-RELATIONSHIP DIAGRAM OF TRANSITTRACK

4.2 ANALYSIS AND NORMALIZATION

Changes from Initial to Final ERD:

- Removed TrainID from ticket and attached it to Route table in the ERD
- Removed stationID from schedule
- Removed DesignationID from Employee table
- Replaced longitude and latitude with StreetAddress and city
- Added status to CustomerSupport

Rationale behind normalization Employee table

It is possible for a new employee type to be added to our system. So, we have normalized the employee table and created a separate table EmployeeType. It is a master data table, so if we add a new employee type, it would be a simple insert operation instead of altering the employee table to modify the check constraint and add another type to the check condition.

The Employee Designation table is also kept as a separate table because an employee can be designated to multiple roles like a technician can work on a train as well as be part of customer service. So, to avoid data redundancy we have performed 2NF normalization for the employee designation.

4.3 FINAL ERD

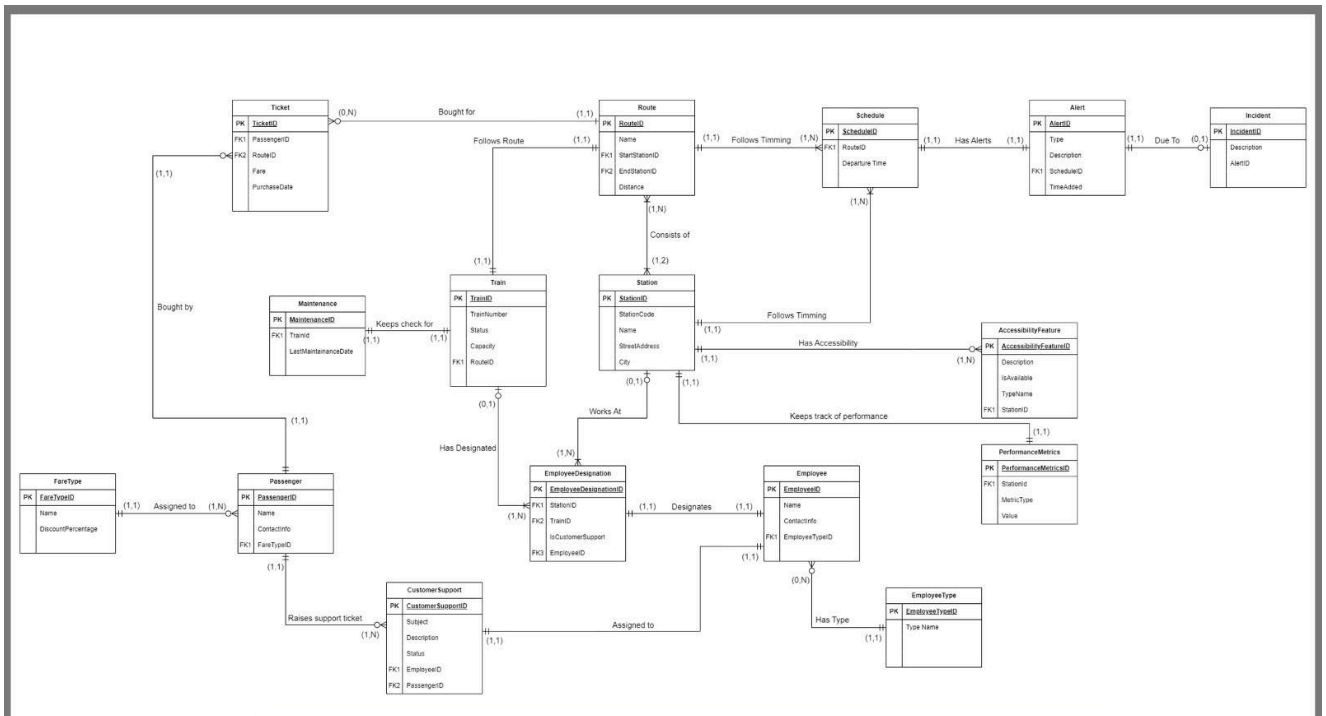


DIAGRAM 4.2. FINAL ENTITY-RELATIONSHIP DIAGRAM OF TRANSITTRACK

5. MAJOR DESIGN ELEMENTS

5.1 ENTITIES

1. Train:

Represents individual trains within the GO Train system. Attributes may include train number, type, capacity, and operational status.

2. Station:

Represents the stations where trains stop. Attributes may include station code, name, street address and city.

3. Route:

Represents the routes followed by trains, including the sequence of stations. Attributes may include route ID, name, starting station, ending station, and distance.

4. Schedule:

Represents the scheduled departure times for trains at each station. Attributes may include schedule ID, route ID, station ID, departure time.

5. Passenger:

Represents individual passengers using the GO Train system. Attributes may include passenger ID, name, contact information, and fare type.

6. Ticket:

Represents tickets purchased by passengers for travel on specific trains. Attributes may include ticket ID, passenger ID, train number, route ID, arrival station, purchase date, and fare.

7. Employee:

Represents employees involved in operating and managing the GO Train system. Attributes may include employee ID, name, employee type, employee designation, and contact information.

8. Maintenance:

Represents maintenance activities performed on trains and keeps the dates of maintenance to keep track of history. Attributes may include maintenance ID, Train ID and Last Maintenance Date.

9. Incident:

Represents incidents or disruptions affecting train operations, such as delays, cancellations, or accidents. Attributes may include incident ID, description, date/time and the alert ID of the alert raised.

10. Fare Type:

Represents the fare type master data for passengers. Passengers can have different tiers of fare type. For example, Adult, Youth, Student etc.

11. Accessibility Feature:

Represents accessibility features available at stations to accommodate passengers with disabilities. Attributes may include feature ID, description, and location.

12. Alert:

Represents alerts or notifications issued to passengers regarding service disruptions, schedule changes, or other important information. Attributes may include alert ID, type, description, and affected schedule ID.

13. Performance Metrics:

Represents key performance indicators (KPIs) used to measure the performance and reliability of the GO Train system, such as on-time performance, average delay duration, and customer satisfaction scores.

14. Customer Support:

Represents the online support tickets raised. Each ticket will have an employee assigned and will contain customer email, subject and description.

5.2 RELATIONSHIPS/ASSOCIATIONS

5.2.1 Train - Route:

- One-to-One relationship: *A train can operate on a single route going back and forth and each route is served by one train.*

5.2.2 Route - Station:

- One-to-Many relationship: *A route consists of multiple stations, and a station can be part of multiple routes.*

5.2.3 Route - Schedule:

- One-to-Many relationship: *A route has multiple schedules for different trains, but each schedule corresponds to one route.*

5.2.4 Train - Maintenance:

- One-to-One relationship: *A train has one maintenance ID, and one maintenance ID keeps track of one train.*

5.2.5 Passenger - Ticket:

- One-to-Many relationship: *A passenger can have multiple tickets for different trains, but each ticket belongs to one passenger.*

5.2.6 Train - Ticket:

- One-to-Many relationship: *A train can have multiple tickets sold for its trips, but each ticket corresponds to one train.*

5.2.7 Station - Schedule:

- One-to-Many relationship: *A station can have multiple schedules for different trains, but each schedule corresponds to one station.*

5.2.8 Train – Employee Designation:

- One-to-Many relationship: *A train has multiple employees designated to work on it, but each employee is assigned to one train.*

5.2.9 Employee Designation - Employee:

- One-to-One relationship: *An employee is designated to one area and each designation is for only employee.*

5.2.10 Incident - Alert:

- One-to-One relationship: *An incident can raise one alert, and one alert can signify only one incident.*

5.2.11 Customer Support - Passenger:

- One-to-Many relationship: *A passenger can raise multiple support tickets, but each support ticket corresponds to one passenger.*

5.2.12 Customer Support - Employee:

- One-to-One relationship: *One ticket is assigned to only one employee and an employee can only work on one ticket at a time.*

5.2.13 Fare Type - Passenger:

- One-to-Many relationship: *A fair type can be assigned to many passengers, but a passenger can only utilize one fare type at a time.*

5.2.14 Station – Accessibility Feature:

- One-to-Many relationship: *A station can have multiple accessibility features, but one accessibility feature signifies only one station.*

5.2.15 Station - Performance Metrics:

- One-to-One relationship: *A station can have one performance metric recorded to keep track of current performance, and each performance metric is for one train*

6. DATABASE SCRIPTS

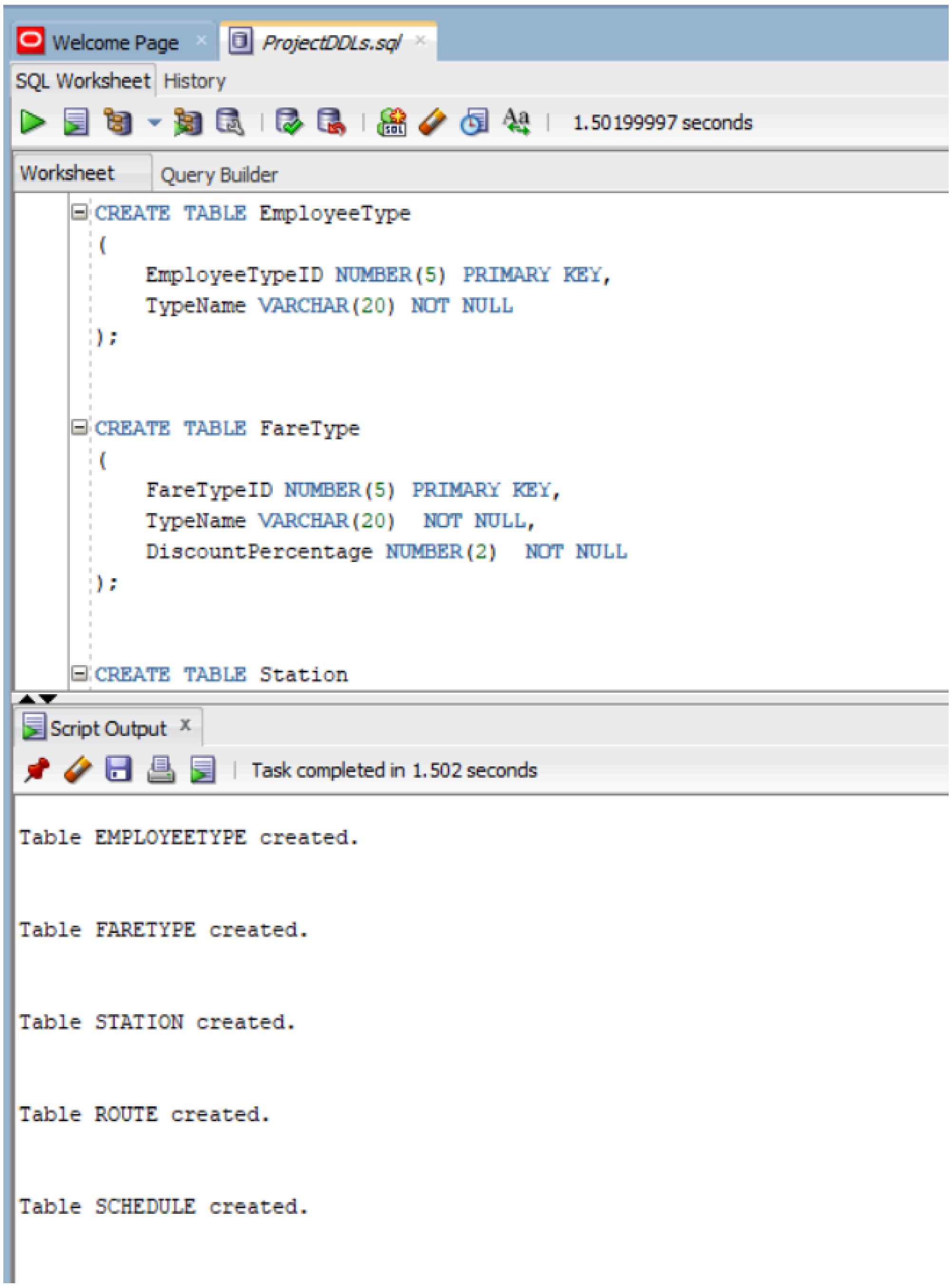
6.1 DDL SCRIPTS

Create Scripts:

DDL script to create tables

```
CREATE TABLE EmployeeType
(
      EmployeeTypeID NUMBER(5) PRIMARY KEY,
      TypeName VARCHAR(20) NOT NULL
);

CREATE TABLE FareType
(
      FareTypeID NUMBER(5) PRIMARY KEY,
      TypeName VARCHAR(20)  NOT NULL,
      DiscountPercentage NUMBER(2)  NOT NULL
);

CREATE TABLE Station
(
	StationID NUMBER(5) PRIMARY KEY,
	StationCode VARCHAR(5) NOT NULL UNIQUE,
	Name VARCHAR(20) NOT NULL,
	StreetAddress VARCHAR(50) NOT NULL,
	City VARCHAR(20)
);

CREATE TABLE Route
(
	RouteID NUMBER(5) PRIMARY KEY,
	StartStationID NUMBER(5) NOT NULL,
	EndStationID NUMBER(5) NOT NULL,
	Distance NUMBER(6),
	CONSTRAINT route_start_station_fk FOREIGN KEY (StartStationID)
	REFERENCES Station(StationID),
	CONSTRAINT route_end_station_fk2 FOREIGN KEY (EndStationID)
	REFERENCES Station(StationID)
);

CREATE TABLE Schedule
(
	ScheduleID NUMBER(5) PRIMARY KEY,
	RouteID NUMBER(5) NOT NULL,
	DepartureTime TIMESTAMP NOT NULL,
	CONSTRAINT schedule_route_fk FOREIGN KEY (RouteID)
	REFERENCES Route(RouteID)
);

CREATE TABLE Train
(
	TrainID NUMBER(5) PRIMARY KEY,
	TrainNumber VARCHAR(5) NOT NULL UNIQUE,
	Status VARCHAR(20) NOT NULL,
	Capacity NUMBER(3),
	RouteId NUMBER(5) NOT NULL,
	CONSTRAINT train_route_fk FOREIGN KEY (RouteId)
	REFERENCES Route(RouteID)
);

CREATE TABLE Employee
(
	EmployeeID NUMBER(5) PRIMARY KEY,
	Name VARCHAR(20) NOT NULL,
	EmployeeTypeID NUMBER(5) NOT NULL,
	ContactInfo VARCHAR(30) NOT NULL,
	CONSTRAINT employee_employeetype_fk FOREIGN KEY (EmployeeTypeID)
	REFERENCES EmployeeType(EmployeeTypeID)
);

CREATE TABLE EmployeeDesignation
(
	EmployeeDesignationID NUMBER(5) PRIMARY KEY,
	EmployeeID NUMBER(5) NOT NULL,
	TrainID NUMBER(5),
	StationID NUMBER(5),
	IsCustomerSupport NUMBER(1) NOT NULL,
	CONSTRAINT employeeDesignation_employee_fk FOREIGN KEY (EmployeeID)
	REFERENCES Employee(EmployeeID),
	CONSTRAINT employeeDesignation_train_fk FOREIGN KEY (TrainID)
	REFERENCES Train(TrainID),
	CONSTRAINT employeeDesignation_station_fk FOREIGN KEY (StationID)
	REFERENCES Station(StationID)
);

CREATE TABLE Alert
(
	AlertID NUMBER(5) PRIMARY KEY,
	Type VARCHAR(20) NOT NULL,
	Description VARCHAR(100),
	ScheduleID NUMBER(5) NOT NULL,
	TimeAdded NUMBER(2),
	CONSTRAINT alert_schedule_fk FOREIGN KEY (ScheduleID)
	REFERENCES Schedule(ScheduleID)
);

CREATE TABLE Incident
(
	IncidentID NUMBER(5) PRIMARY KEY,
	Description VARCHAR(100) NOT NULL,
	AlertID NUMBER(5) NOT NULL,
	CONSTRAINT incident_alert_fk FOREIGN KEY (AlertID)
	REFERENCES Alert(AlertID)
```

```
);

CREATE TABLE Passenger
(
      PassengerID NUMBER(5) PRIMARY KEY,
      Name VARCHAR(20) NOT NULL,
      ContactInfo VARCHAR(30) NOT NULL,
      FareTypeID NUMBER(5) NOT NULL,
      CONSTRAINT passenger_faretype_fk FOREIGN KEY (FareTypeID)
      REFERENCES FareType(FareTypeID)
);

CREATE TABLE Ticket
(
      TicketID NUMBER(5) PRIMARY KEY,
      RouteID NUMBER(5) NOT NULL,
      PassengerID NUMBER(5),
      Fare NUMBER(3) NOT NULL,
      PurchaseTime TIMESTAMP DEFAULT SYSTIMESTAMP,
      CONSTRAINT ticket_route_fk FOREIGN KEY (RouteID)
      REFERENCES Route(RouteID),
      CONSTRAINT ticket_passenger_fk FOREIGN KEY (PassengerID)
      REFERENCES Passenger(PassengerID)
);

CREATE TABLE CustomerSupport
(
      CustomerSupportID NUMBER(5) PRIMARY KEY,
      Status VARCHAR(20) NOT NULL,
      Subject VARCHAR(50),
      Description VARCHAR(200) NOT NULL,
      PassengerID NUMBER(5) NOT NULL,
      EmployeeID NUMBER(5) NOT NULL,
      CONSTRAINT support_employee_fk FOREIGN KEY (EmployeeID)
      REFERENCES Employee(EmployeeID),
      CONSTRAINT support_passenger_fk FOREIGN KEY (PassengerID)
      REFERENCES Passenger(PassengerID)
);

CREATE TABLE Maintenance
(
      MaintenanceID NUMBER(5) PRIMARY KEY,
      TrainID NUMBER(5) NOT NULL,
      LastMaintainanceDate DATE NOT NULL,
      CONSTRAINT maintenance_train_fk FOREIGN KEY (TrainID)
      REFERENCES Train(TrainID)
);

CREATE TABLE PerformanceMetrics
(
      PerformanceMetricsID NUMBER(5) PRIMARY KEY,
      MetricType VARCHAR(100) NOT NULL,
      Value NUMBER(2) NOT NULL,
      StationID NUMBER(5) NOT NULL,
      CONSTRAINT performance_station_fk FOREIGN KEY (StationID)
      REFERENCES Station(StationID)
);

CREATE TABLE AccessibilityFeature
(
      AccessibilityFeatureID NUMBER(5) PRIMARY KEY,
      TypeName VARCHAR(20) NOT NULL,
      IsAvailable NUMBER(1) NOT NULL,
      StationID NUMBER(5) NOT NULL,
      CONSTRAINT accessibility_station_fk FOREIGN KEY (StationID)
      REFERENCES Station(StationID)
);
```



```
CREATE TABLE EmployeeType
(
    EmployeeTypeID NUMBER(5) PRIMARY KEY,
    TypeName VARCHAR(20) NOT NULL
);


CREATE TABLE FareType
(
    FareTypeID NUMBER(5) PRIMARY KEY,
    TypeName VARCHAR(20)  NOT NULL,
    DiscountPercentage NUMBER(2)  NOT NULL
);


CREATE TABLE Station
```

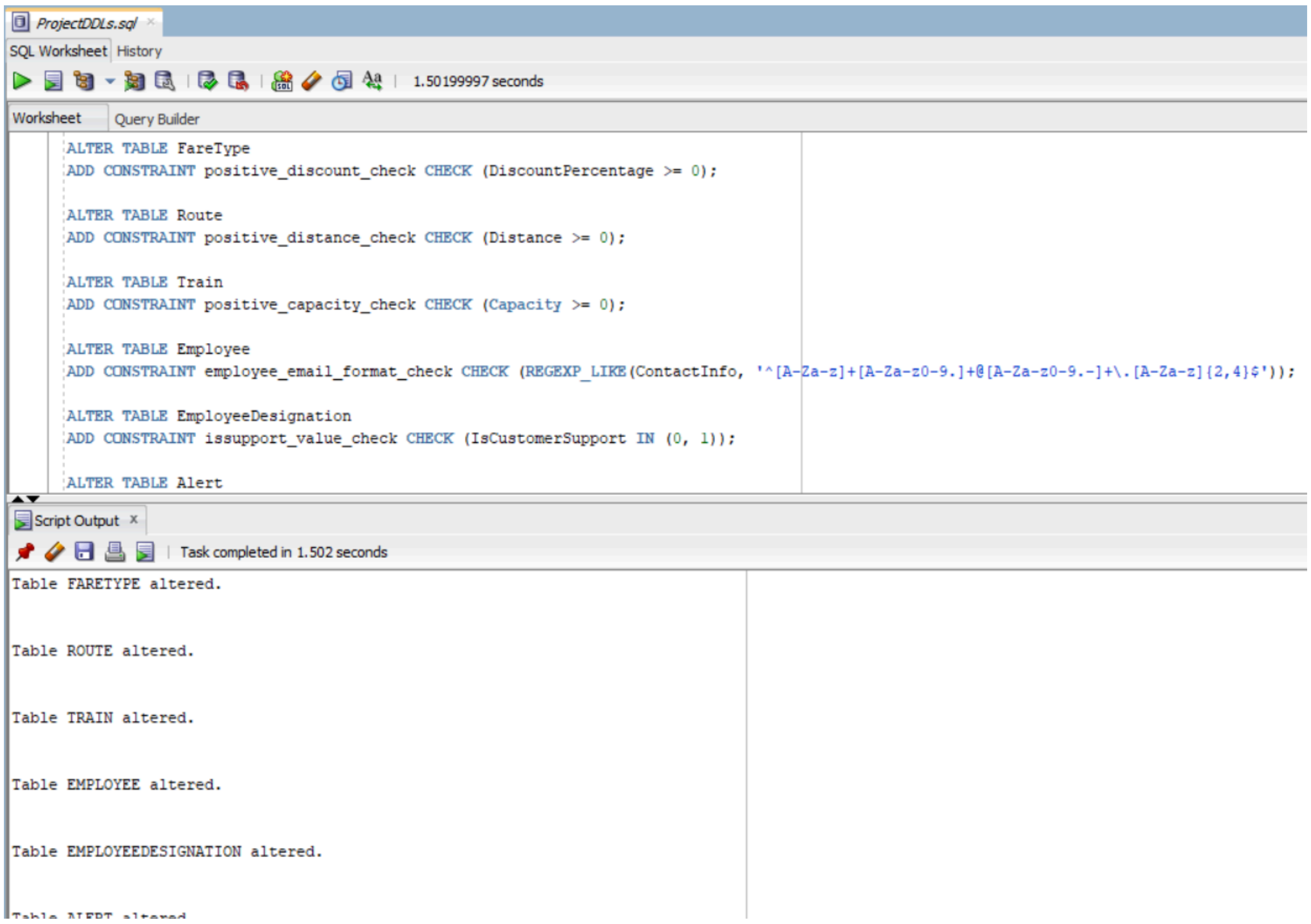
```
Table EMPLOYEETYPE created.


Table FARETYPE created.


Table STATION created.


Table ROUTE created.


Table SCHEDULE created.
```

Alter Scripts:

DDL script to alter tables for adding additional constraints.

```
ALTER TABLE FareType
ADD CONSTRAINT positive_discount_check CHECK (DiscountPercentage >= 0);

ALTER TABLE Route
ADD CONSTRAINT positive_distance_check CHECK (Distance >= 0);

ALTER TABLE Train
```

```
ADD CONSTRAINT positive_capacity_check CHECK (Capacity >= 0);

ALTER TABLE Employee
ADD CONSTRAINT employee_email_format_check CHECK (REGEXP_LIKE(ContactInfo,
'^[A-Za-z]+[A-Za-z0-9.]+@[A-Za-z0-9.-]+¥.[A-Za-z]{2,4}$'));

ALTER TABLE EmployeeDesignation
ADD CONSTRAINT issupport_value_check CHECK (IsCustomerSupport IN (0, 1));

ALTER TABLE Alert
ADD CONSTRAINT alert_type_value_check CHECK (Type IN ('Incident', 'Construction',
'Late Arrival', 'Other'));

ALTER TABLE Alert
ADD CONSTRAINT positive_time_added_check CHECK (TimeAdded >= 0);

ALTER TABLE Passenger
ADD CONSTRAINT passenger_email_format_check CHECK (REGEXP_LIKE(ContactInfo,
'^[A-Za-z]+[A-Za-z0-9.]+@[A-Za-z0-9.-]+¥.[A-Za-z]{2,4}$'));

ALTER TABLE Ticket
ADD CONSTRAINT positive_fare_check CHECK (Fare >= 0);

ALTER TABLE CustomerSupport
ADD CONSTRAINT support_status_value_check CHECK (Status IN ('Open', 'Closed'));

ALTER TABLE PerformanceMetrics
ADD CONSTRAINT metric_type_value_check CHECK (MetricType IN ('Cleaniness',
'Railcar Delays'));

ALTER TABLE PerformanceMetrics
ADD CONSTRAINT positive_metric_value_check CHECK (Value >= 0);

ALTER TABLE AccessibilityFeature
ADD CONSTRAINT accessibility_type_value_check CHECK (TypeName IN ('Wheel Chair',
'Parking', 'Accessible washrooms'));

ALTER TABLE AccessibilityFeature
ADD CONSTRAINT isavailable_value_check CHECK (IsAvailable IN (0, 1));
```



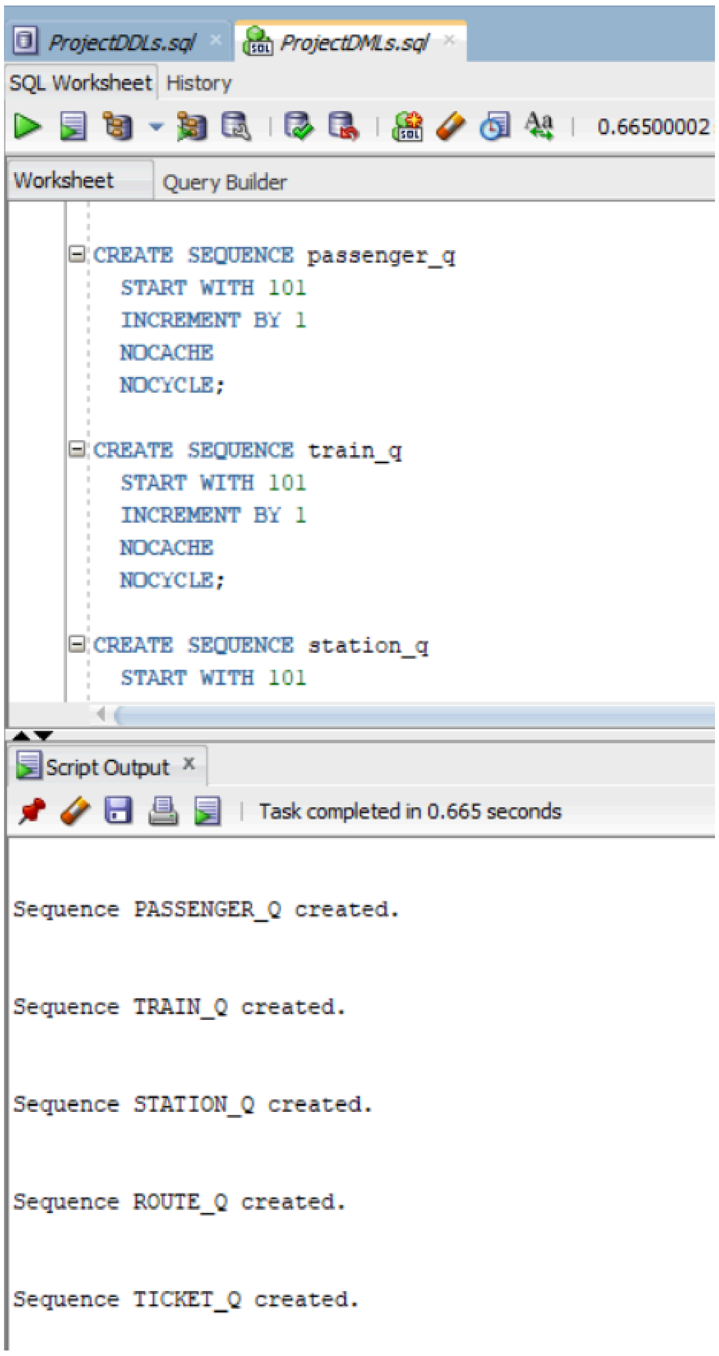
```
ALTER TABLE FareType
ADD CONSTRAINT positive_discount_check CHECK (DiscountPercentage >= 0);

ALTER TABLE Route
ADD CONSTRAINT positive_distance_check CHECK (Distance >= 0);

ALTER TABLE Train
ADD CONSTRAINT positive_capacity_check CHECK (Capacity >= 0);

ALTER TABLE Employee
ADD CONSTRAINT employee_email_format_check CHECK (REGEXP_LIKE(ContactInfo, '^[A-Za-z]+[A-Za-z0-9.]+@[A-Za-z0-9.-]+\.[A-Za-z]{2,4}$'));

ALTER TABLE EmployeeDesignation
ADD CONSTRAINT issupport_value_check CHECK (IsCustomerSupport IN (0, 1));

ALTER TABLE Alert
```

```
Table FARETYPE altered.


Table ROUTE altered.


Table TRAIN altered.


Table EMPLOYEE altered.


Table EMPLOYEEDESIGNATION altered.


Table ALERT altered.
```

Create Sequence Scripts:

DDL script to create sequences for the tables

```
CREATE SEQUENCE passenger_q
  START WITH 101
  INCREMENT BY 1
  NOCACHE
  NOCYCLE;

CREATE SEQUENCE train_q
  START WITH 101
  INCREMENT BY 1
  NOCACHE
  NOCYCLE;

CREATE SEQUENCE station_q
  START WITH 101
  INCREMENT BY 1
  NOCACHE
  NOCYCLE;

CREATE SEQUENCE route_q
  START WITH 1
  INCREMENT BY 1
  NOCACHE
  NOCYCLE;

CREATE SEQUENCE ticket_q
  START WITH 1
  INCREMENT BY 1
  NOCACHE
  NOCYCLE;

CREATE SEQUENCE schedule_q
  START WITH 1
  INCREMENT BY 1
  NOCACHE
  NOCYCLE;

CREATE SEQUENCE alert_q
  START WITH 1
  INCREMENT BY 1
  NOCACHE
  NOCYCLE;

CREATE SEQUENCE support_q
  START WITH 1
  INCREMENT BY 1
  NOCACHE
  NOCYCLE;
```



```
CREATE SEQUENCE passenger_q
  START WITH 101
  INCREMENT BY 1
  NOCACHE
  NOCYCLE;

CREATE SEQUENCE train_q
  START WITH 101
  INCREMENT BY 1
  NOCACHE
  NOCYCLE;

CREATE SEQUENCE station_q
  START WITH 101
```

```
Sequence PASSENGER_Q created.


Sequence TRAIN_Q created.


Sequence STATION_Q created.


Sequence ROUTE_Q created.


Sequence TICKET_Q created.
```

6.2 DML SCRIPTS

- **Insert Scripts:**

One query of each insert operation

- **EmployeeType:**

INSERT INTO EmployeeType (EmployeeTypeID, TypeName) VALUES (1, ’Engine Driver’);

- **FareType:**

INSERT INTO FareType (FareTypeID, TypeName, DiscountPercentage) VALUES (1, ’Normal’, 0);

- **Station:**

```
INSERT INTO Station (StationID, StationCode, Name, StreetAddress, City) VALUES (station_q.nextval, 'OAKTT', 'Oakville Station', '2080 Nicholle Crossing', 'Oakville');
```

- **Route:**

```
INSERT INTO Route (RouteID, StartStationID, EndStationID, Distance) VALUES (route_q.nextval, 101, 103, 481);
```

- **Schedule:**

```
INSERT INTO Schedule (ScheduleID, RouteID, DepartureTime) VALUES (schedule_q.nextval, 2, '13-APR-24 08.00.00.000000000');
```

- **Train:**

```
INSERT INTO Train (TrainID, TrainNumber, Status, Capacity, RouteId) VALUES (train_q.nextval, '21A', 'In Service', 80, 3);
```

- **Employee:**

```
INSERT INTO Employee (EmployeeID, Name, EmployeeTypeID, ContactInfo) VALUES (1, 'Gail Richardson', 2, 'g.richardson@gmail.com');
```

- **EmployeeDesignation:**

```
INSERT INTO EmployeeDesignation (EmployeeDesignationID, EmployeeID, TrainID, StationID, IsCustomerSupport) VALUES (1, 1, null, 101, 0);
```

- **Alert:**

```
INSERT INTO Alert (AlertID, Type, Description, ScheduleID, TimeAdded) VALUES (alert_q.nextval, 'Incident', 'Incident', 2, 15);
```

- **Incident:**

```
INSERT INTO Incident (IncidentID, Description, AlertID) VALUES (1, 'Deer stuck on train tracks', 1);
```

- **Passenger:**

```
INSERT INTO Passenger (PassengerID, Name, ContactInfo, FareTypeID) VALUES (passenger_q.nextval, 'Vanessa Wade', 'v.wade@gmail.com', 5);
```

- **Ticket:**

```
INSERT INTO Ticket (TicketID, RouteID, PassengerID, Fare) VALUES (ticket_q.nextval, 2, 101, 7);
```

- **CustomerSupport:**

```
INSERT INTO CustomerSupport (CustomerSupportID, Status, Subject, Description, PassengerID, EmployeeID) VALUES (support_q.nextval, 'Open', 'Refund Ticket', 'Hi, I have accidently bought the wrong ticket can I refund it please', 102, 5);
```

- **Maintenance:**

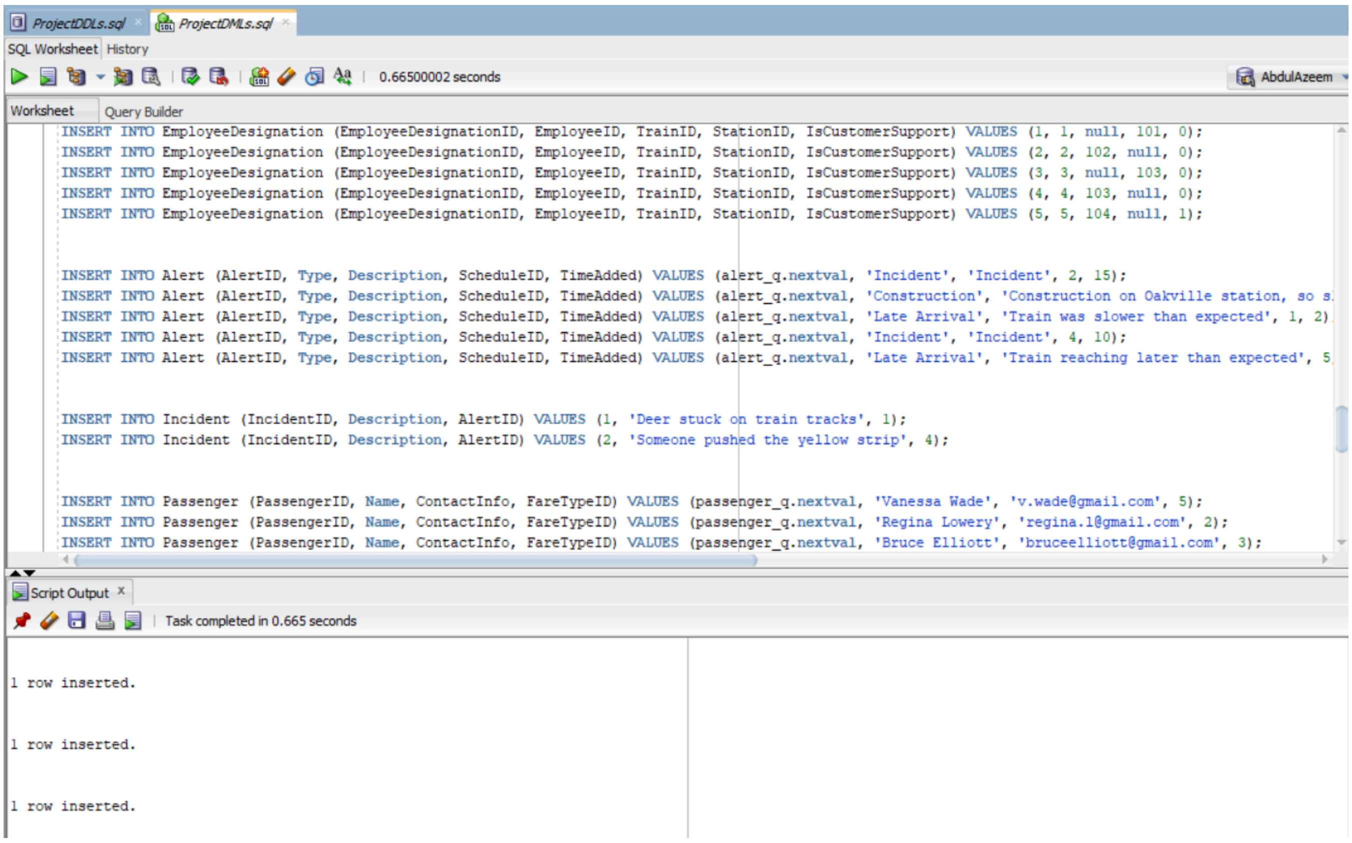
```
INSERT INTO Maintenance (MaintenanceID, TrainID, LastMaintainanceDate) VALUES (1, 101, '13-APR-24');
```

- **PerformanceMetrics:**

```
INSERT INTO PerformanceMetrics (PerformanceMetricsID, MetricType, Value, StationID) VALUES (1, 'Cleaniness', 7, 102);
```

- **AccessibilityFeature:**

```
INSERT INTO AccessibilityFeature (AccessibilityFeatureID, TypeName, IsAvailable, StationID) VALUES (1, 'Wheelchair', 1, 102);
```



```
INSERT INTO EmployeeDesignation (EmployeeDesignationID, EmployeeID, TrainID, StationID, IsCustomerSupport) VALUES (1, 1, null, 101, 0);
INSERT INTO EmployeeDesignation (EmployeeDesignationID, EmployeeID, TrainID, StationID, IsCustomerSupport) VALUES (2, 2, 102, null, 0);
INSERT INTO EmployeeDesignation (EmployeeDesignationID, EmployeeID, TrainID, StationID, IsCustomerSupport) VALUES (3, 3, null, 103, 0);
INSERT INTO EmployeeDesignation (EmployeeDesignationID, EmployeeID, TrainID, StationID, IsCustomerSupport) VALUES (4, 4, 103, null, 0);
INSERT INTO EmployeeDesignation (EmployeeDesignationID, EmployeeID, TrainID, StationID, IsCustomerSupport) VALUES (5, 5, 104, null, 1);


INSERT INTO Alert (AlertID, Type, Description, ScheduleID, TimeAdded) VALUES (alert_q.nextval, 'Incident', 'Incident', 2, 15);
INSERT INTO Alert (AlertID, Type, Description, ScheduleID, TimeAdded) VALUES (alert_q.nextval, 'Construction', 'Construction on Oakville station, so s
INSERT INTO Alert (AlertID, Type, Description, ScheduleID, TimeAdded) VALUES (alert_q.nextval, 'Late Arrival', 'Train was slower than expected', 1, 2)
INSERT INTO Alert (AlertID, Type, Description, ScheduleID, TimeAdded) VALUES (alert_q.nextval, 'Incident', 'Incident', 4, 10);
INSERT INTO Alert (AlertID, Type, Description, ScheduleID, TimeAdded) VALUES (alert_q.nextval, 'Late Arrival', 'Train reaching later than expected', 5


INSERT INTO Incident (IncidentID, Description, AlertID) VALUES (1, 'Deer stuck on train tracks', 1);
INSERT INTO Incident (IncidentID, Description, AlertID) VALUES (2, 'Someone pushed the yellow strip', 4);


INSERT INTO Passenger (PassengerID, Name, ContactInfo, FareTypeID) VALUES (passenger_q.nextval, 'Vanessa Wade', 'v.wade@gmail.com', 5);
INSERT INTO Passenger (PassengerID, Name, ContactInfo, FareTypeID) VALUES (passenger_q.nextval, 'Regina Lowery', 'regina.l@gmail.com', 2);
INSERT INTO Passenger (PassengerID, Name, ContactInfo, FareTypeID) VALUES (passenger_q.nextval, 'Bruce Elliott', 'bruceelliott@gmail.com', 3);
```

```
1 row inserted.

1 row inserted.

1 row inserted.
```

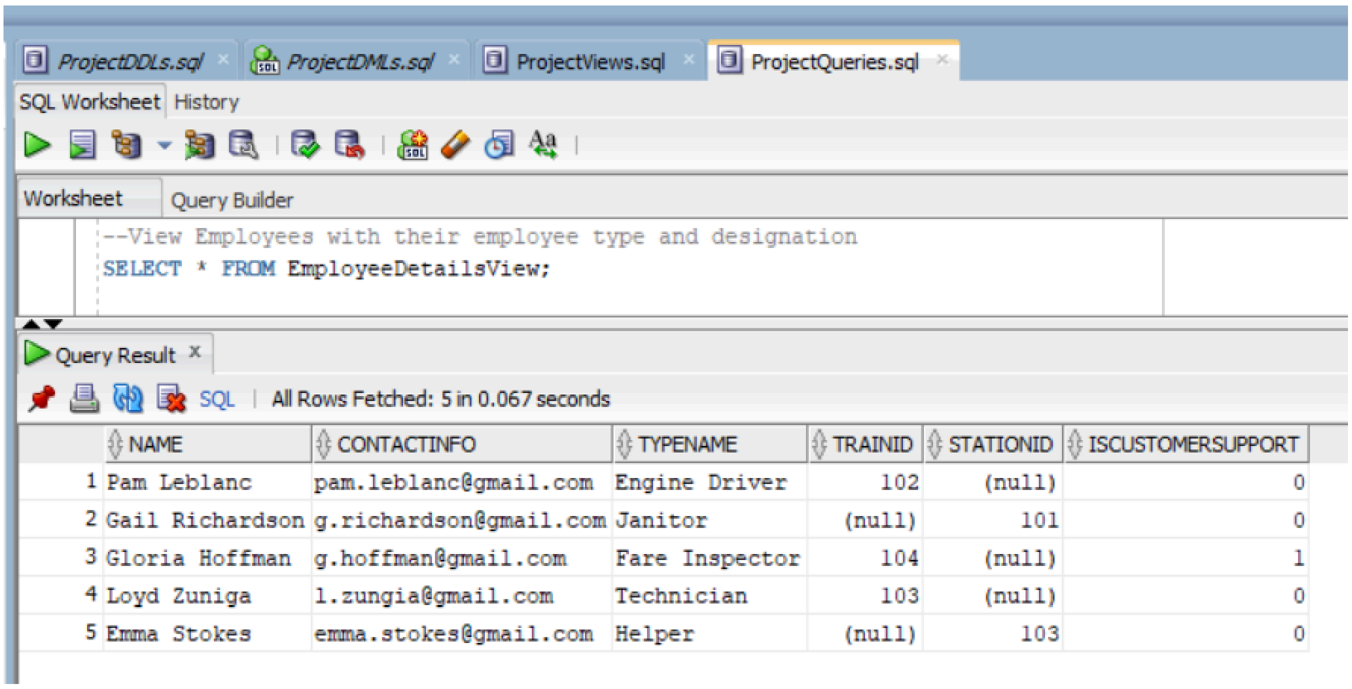
6.3 VIEWS

6.3.1 EmployeeDetailsView

Description: View Employees with their employee type and designation

Query:

```
CREATE VIEW EmployeeDetailsView AS
SELECT e.Name, e.ContactInfo, et.TypeName, ed.TrainID, ed.StationID,
ed.IsCustomerSupport
FROM Employee e
INNER JOIN EmployeeType et ON et.EmployeeTypeID = e.EmployeeTypeID
INNER JOIN EmployeeDesignation ed ON ed.EmployeeID = e.EmployeeID;
```



```
--View Employees with their employee type and designation
SELECT * FROM EmployeeDetailsView;
```

All Rows Fetched: 5 in 0.067 seconds

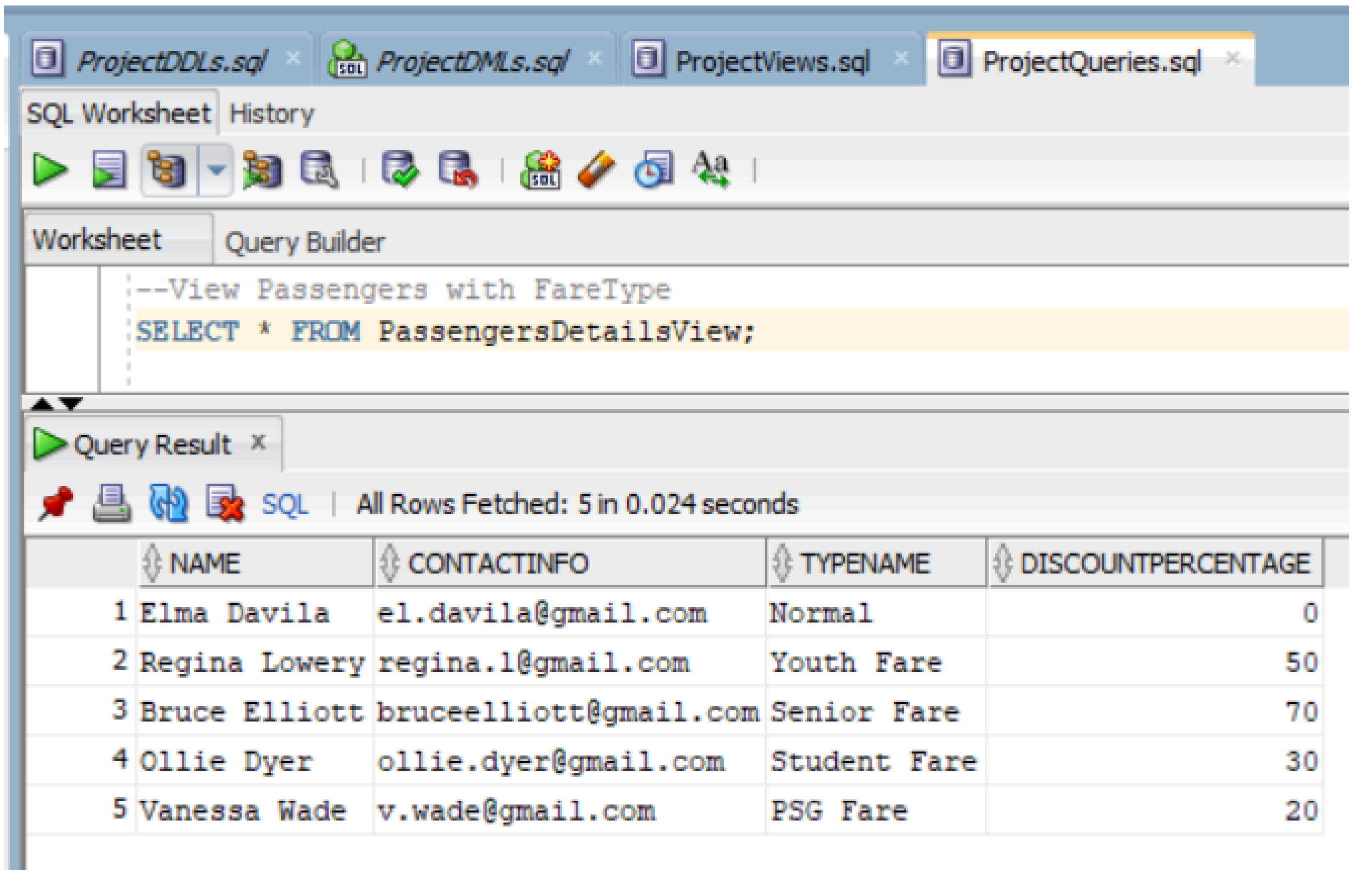
```
  NAME             CONTACTINFO                TYPENAME         TRAINID  STATIONID  ISCUSTOMERSUPPORT
1 Pam Leblanc      pam.leblanc@gmail.com      Engine Driver        102     (null)                  0
2 Gail Richardson  g.richardson@gmail.com     Janitor           (null)        101                  0
3 Gloria Hoffman   g.hoffman@gmail.com        Fare Inspector       104     (null)                  1
4 Loyd Zuniga      l.zungia@gmail.com         Technician           103     (null)                  0
5 Emma Stokes      emma.stokes@gmail.com      Helper            (null)        103                  0
```

6.3.2 PassengersDetailsView

Description: View Passengers with FareType

Query:

```
CREATE VIEW PassengersDetailsView AS
SELECT p.Name, p.ContactInfo, ft.TypeName, ft.DiscountPercentage
FROM Passenger p
INNER JOIN FareType ft ON ft.FareTypeID = p.FareTypeID;
```



```
--View Passengers with FareType
SELECT * FROM PassengersDetailsView;
```

All Rows Fetched: 5 in 0.024 seconds

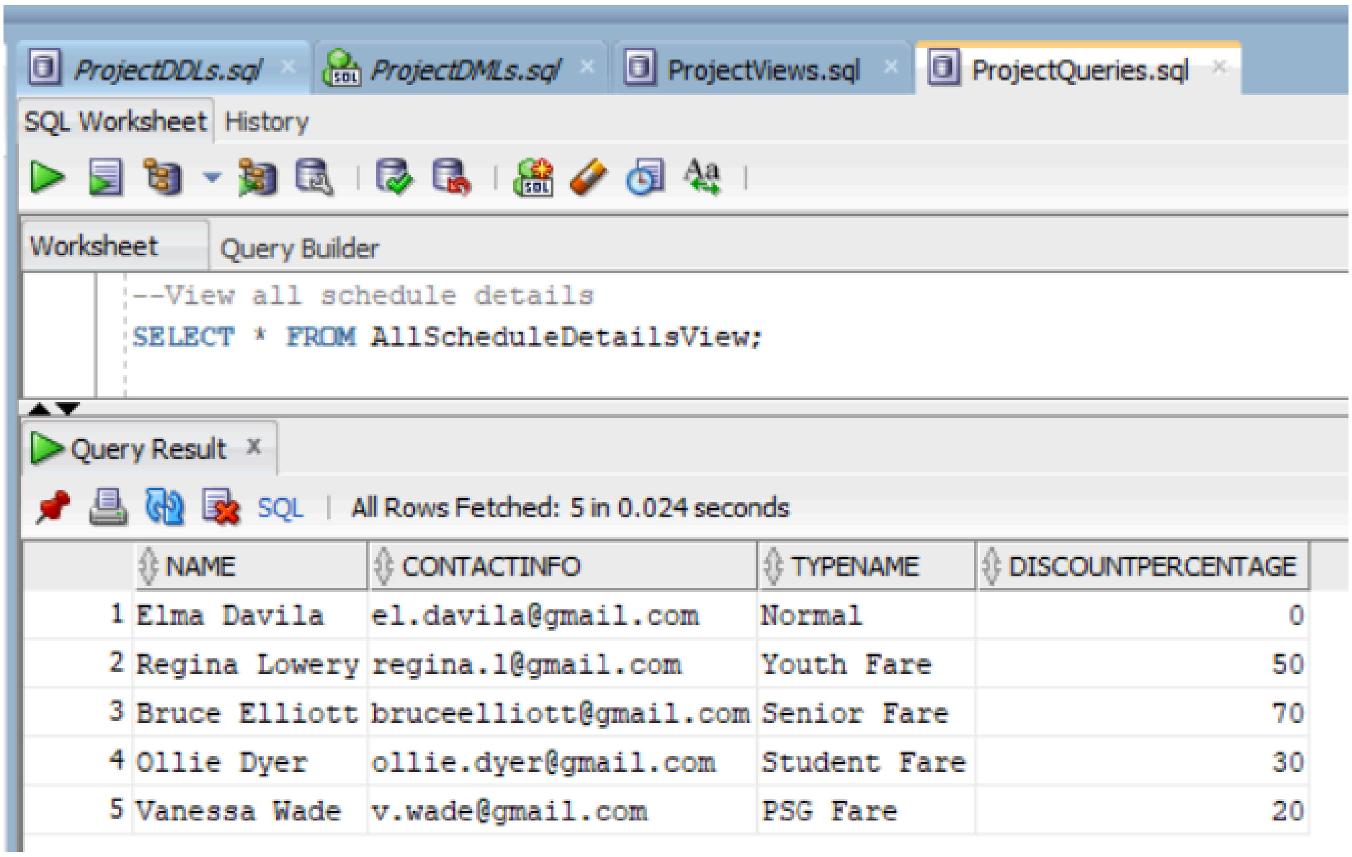
	NAME	CONTACTINFO	TYPENAME	DISCOUNTPERCENTAGE
1	Elma Davila	el.davila@gmail.com	Normal	0
2	Regina Lowery	regina.l@gmail.com	Youth Fare	50
3	Bruce Elliott	bruceelliott@gmail.com	Senior Fare	70
4	Ollie Dyer	ollie.dyer@gmail.com	Student Fare	30
5	Vanessa Wade	v.wade@gmail.com	PSG Fare	20

6.3.3 AllScheduleDetailsView

Description: View all schedule details

Query:

```
CREATE VIEW AllScheduleDetailsView AS
SELECT  s.ScheduleID,  s.DepartureTime,  st1.Name  AS  StartStation,  st2.Name  AS
EndStation
FROM Schedule s
INNER JOIN Route r ON r.RouteID = s.RouteID
LEFT JOIN Station st1 ON st1.StationID = r.StartStationID
LEFT JOIN Station st2 ON st2.StationID = r.EndStationID
ORDER BY s.ScheduleID;
```



```
--View all schedule details
SELECT * FROM AllScheduleDetailsView;
```

All Rows Fetched: 5 in 0.024 seconds

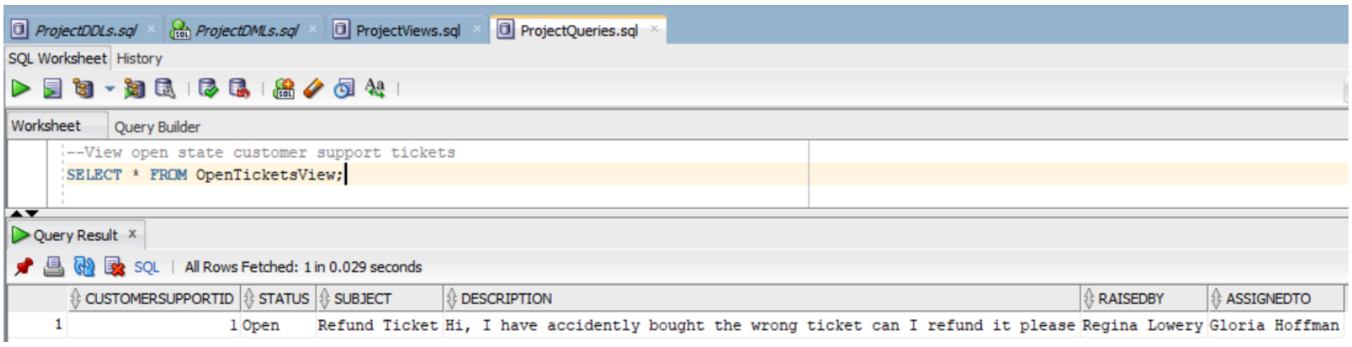
	NAME	CONTACTINFO	TYPENAME	DISCOUNTPERCENTAGE
1	Elma Davila	el.davila@gmail.com	Normal	0
2	Regina Lowery	regina.l@gmail.com	Youth Fare	50
3	Bruce Elliott	bruceelliott@gmail.com	Senior Fare	70
4	Ollie Dyer	ollie.dyer@gmail.com	Student Fare	30
5	Vanessa Wade	v.wade@gmail.com	PSG Fare	20

6.3.4 OpenTicketsView

Description: View open state customer support tickets

Query:

```
CREATE VIEW OpenTicketsView AS
SELECT cs.CustomerSupportID, cs.Status, cs.Subject, cs.Description, p.Name AS
RaisedBy, e.Name AS AssignedTo
FROM CustomerSupport cs
INNER JOIN Passenger p ON p.PassengerID = cs.PassengerID
INNER JOIN Employee e ON e.EmployeeID = cs.EmployeeID
WHERE Status = 'Open';
```



```
--View open state customer support tickets
SELECT * FROM OpenTicketsView;
```

All Rows Fetched: 1 in 0.029 seconds

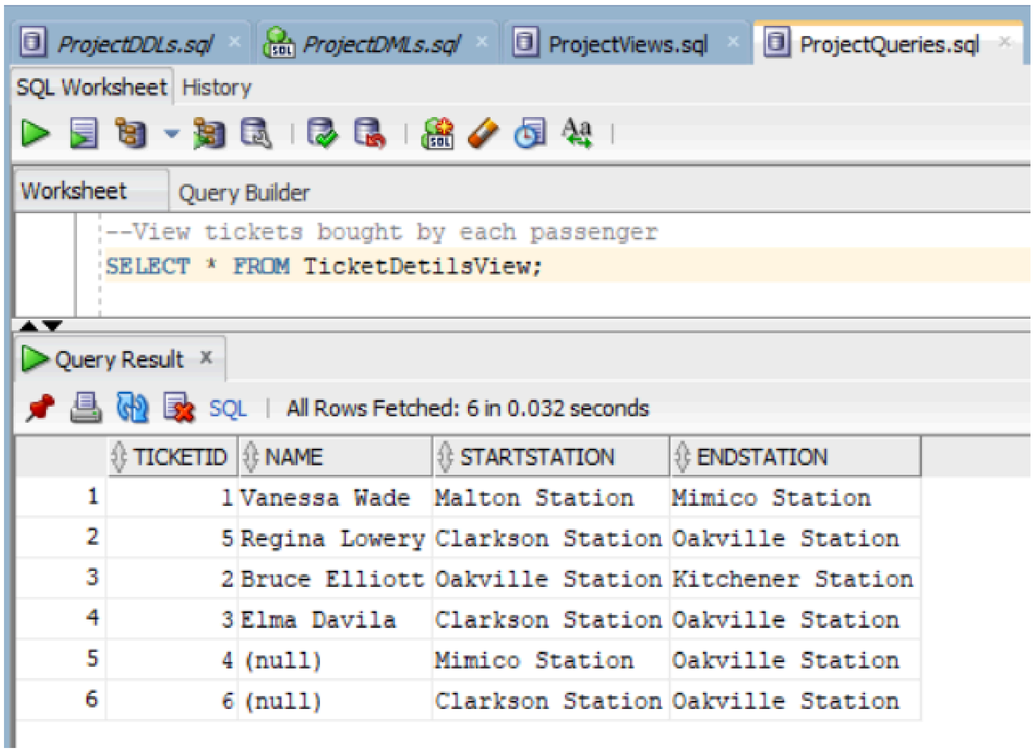
	CUSTOMERSUPPORTID	STATUS	SUBJECT	DESCRIPTION	RAISEDBY	ASSIGNEDTO
1	1	Open	Refund Ticket	Hi, I have accidently bought the wrong ticket can I refund it please	Regina Lowery	Gloria Hoffman

6.3.5 TicketDetilsView

Description: View tickets bought by each passenger

Query:

```
CREATE VIEW TicketDetilsView AS
SELECT t.TicketID, p.Name, st1.Name AS StartStation, st2.Name AS EndStation
FROM Ticket t
LEFT JOIN Passenger p ON p.PassengerID = t.PassengerID
INNER JOIN Route r ON r.RouteID = t.RouteID
LEFT JOIN Station st1 ON st1.StationID = r.StartStationID
LEFT JOIN Station st2 ON st2.StationID = r.EndStationID;
```



```
--View tickets bought by each passenger
SELECT * FROM TicketDetilsView;
```

All Rows Fetched: 6 in 0.032 seconds

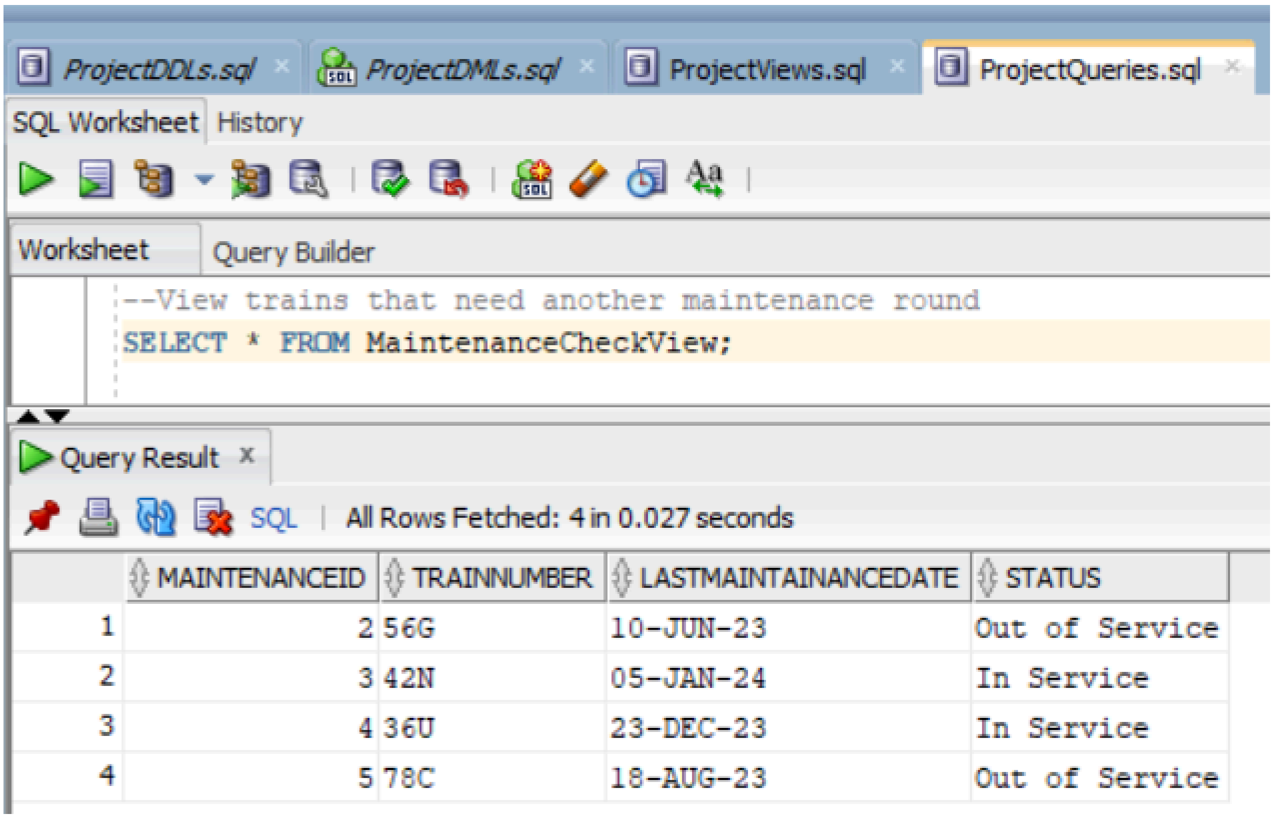
	TICKETID	NAME	STARTSTATION	ENDSTATION
1	1	Vanessa Wade	Malton Station	Mimico Station
2	5	Regina Lowery	Clarkson Station	Oakville Station
3	2	Bruce Elliott	Oakville Station	Kitchener Station
4	3	Elma Davila	Clarkson Station	Oakville Station
5	4	(null)	Mimico Station	Oakville Station
6	6	(null)	Clarkson Station	Oakville Station

6.3.6 MaintenanceCheckView

Description: View trains that need another maintenance round

Query:

```
CREATE VIEW MaintenanceCheckView AS
SELECT m.MaintenanceID, t.TrainNumber, m.LastMaintainanceDate, t.Status
FROM Maintenance m
INNER JOIN Train t ON t.TrainID = m.TrainID
WHERE m.LastMaintainanceDate < SYSDATE - 30;
```



```
--View trains that need another maintenance round
SELECT * FROM MaintenanceCheckView;
```

All Rows Fetched: 4 in 0.027 seconds

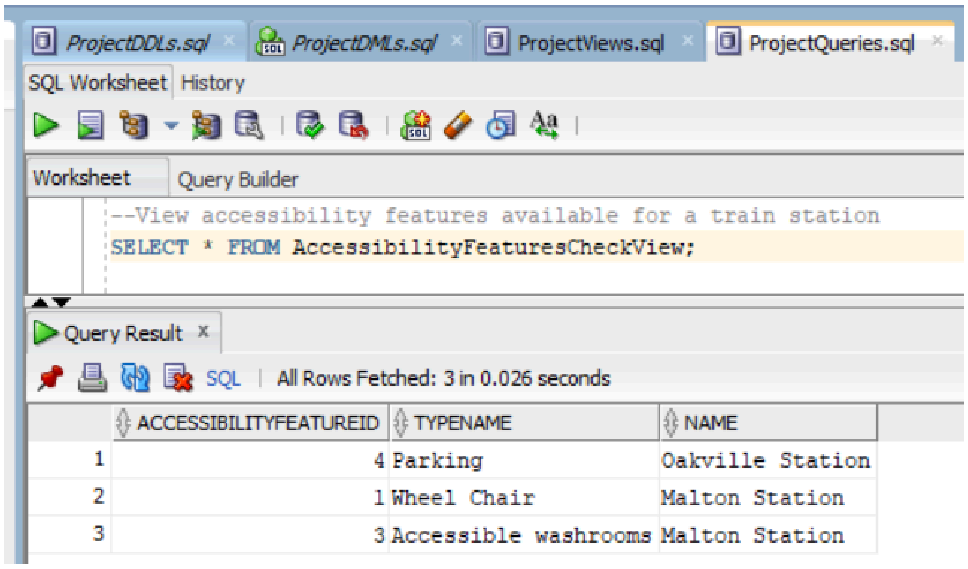
	MAINTENANCEID	TRAINNUMBER	LASTMAINTAINANCEDATE	STATUS
1	2	56G	10-JUN-23	Out of Service
2	3	42N	05-JAN-24	In Service
3	4	36U	23-DEC-23	In Service
4	5	78C	18-AUG-23	Out of Service

6.3.7 AccessibilityFeaturesCheckView

Description: View accessibility features available for a train station

Query:

```
CREATE VIEW AccessibilityFeaturesCheckView AS
SELECT af.AccessibilityFeatureID, af.TypeName, s.Name
FROM AccessibilityFeature af
INNER JOIN Station s ON s.StationID = af.StationID
WHERE af.IsAvailable = 1;
```



```
--View accessibility features available for a train station
SELECT * FROM AccessibilityFeaturesCheckView;
```

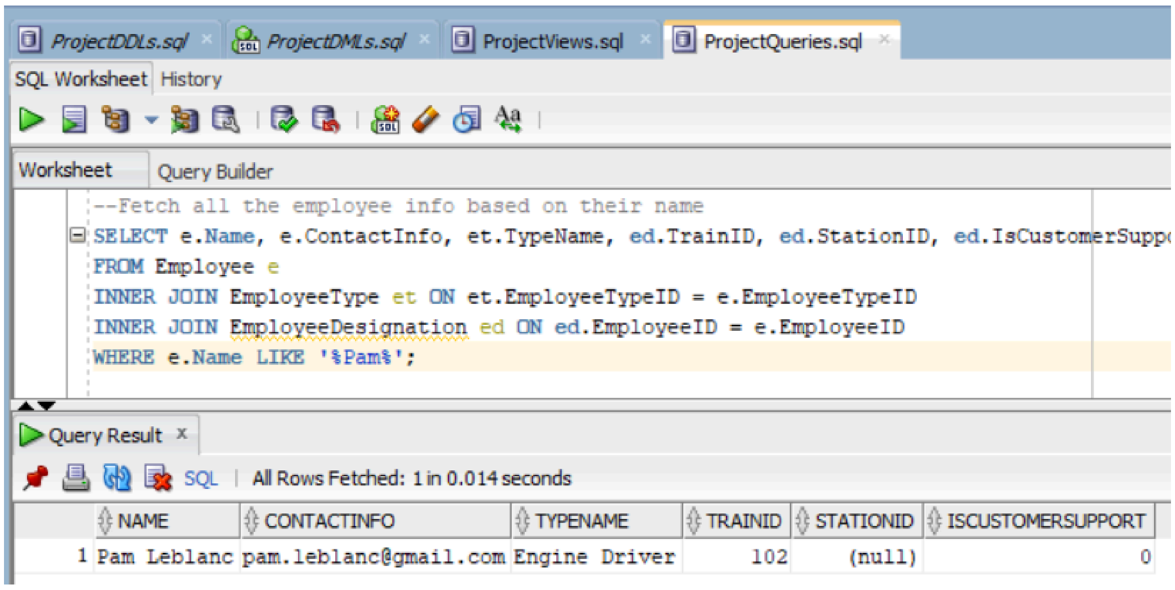
All Rows Fetched: 3 in 0.026 seconds

	ACCESSIBILITYFEATUREID	TYPENAME	NAME
1	4	Parking	Oakville Station
2	1	Wheel Chair	Malton Station
3	3	Accessible washrooms	Malton Station

6.4 SAMPLE TEST QUERIES

6.4.1 Fetch all the employee info based on their name

```
SELECT e.Name, e.ContactInfo, et.TypeName, ed.TrainID, ed.StationID,
ed.IsCustomerSupport
FROM Employee e
INNER JOIN EmployeeType et ON et.EmployeeTypeID = e.EmployeeTypeID
INNER JOIN EmployeeDesignation ed ON ed.EmployeeID = e.EmployeeID
WHERE e.Name LIKE '%Pam%';
```



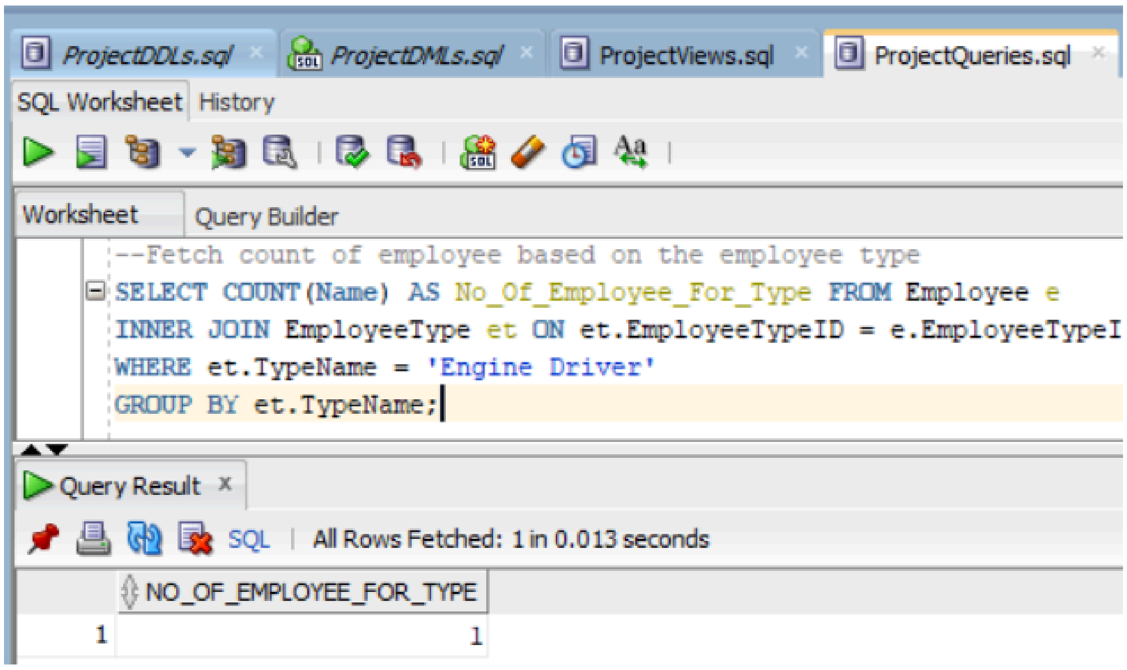
```
--Fetch all the employee info based on their name
SELECT e.Name, e.ContactInfo, et.TypeName, ed.TrainID, ed.StationID, ed.IsCustomerSupp
FROM Employee e
INNER JOIN EmployeeType et ON et.EmployeeTypeID = e.EmployeeTypeID
INNER JOIN EmployeeDesignation ed ON ed.EmployeeID = e.EmployeeID
WHERE e.Name LIKE '%Pam%';
```

All Rows Fetched: 1 in 0.014 seconds

```
  NAME        CONTACTINFO          TYPENAME       TRAINID  STATIONID  ISCUSTOMERSUPPORT
1 Pam Leblanc pam.leblanc@gmail.com Engine Driver     102     (null)                  0
```

6.4.2 Fetch count of employee based on the employee type

```
SELECT COUNT(Name) AS No_Of_Employee_For_Type FROM Employee e
INNER JOIN EmployeeType et ON et.EmployeeTypeID = e.EmployeeTypeID
WHERE et.TypeName = 'Engine Driver'
GROUP BY et.TypeName;
```



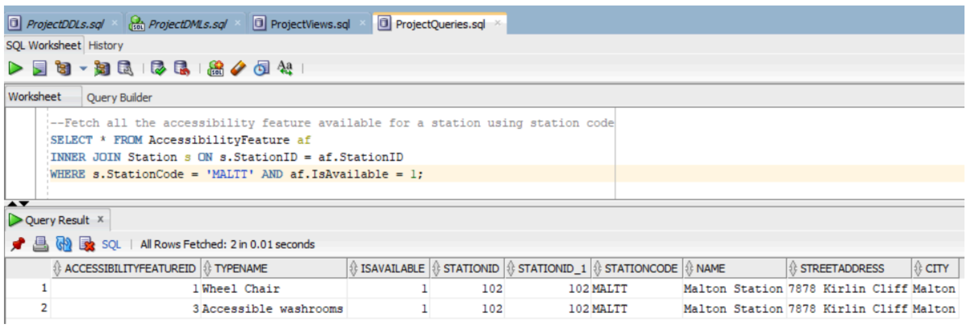
```
--Fetch count of employee based on the employee type
SELECT COUNT(Name) AS No_Of_Employee_For_Type FROM Employee e
INNER JOIN EmployeeType et ON et.EmployeeTypeID = e.EmployeeTypeI
WHERE et.TypeName = 'Engine Driver'
GROUP BY et.TypeName;
```

All Rows Fetched: 1 in 0.013 seconds

	NO_OF_EMPLOYEE_FOR_TYPE
1	1

6.4.3 Fetch all the accessibility feature available for a station using station code

```
SELECT * FROM AccessibilityFeature af
INNER JOIN Station s ON s.StationID = af.StationID
WHERE s.StationCode = ’MALTT’ AND af.IsAvailable = 1;
```



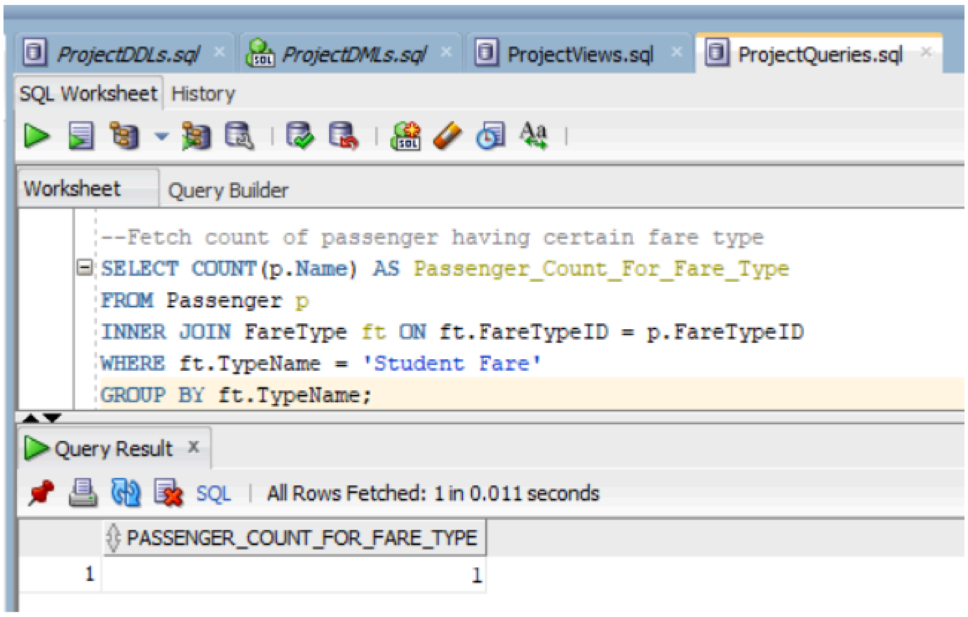
```
--Fetch all the accessibility feature available for a station using station code
SELECT * FROM AccessibilityFeature af
INNER JOIN Station s ON s.StationID = af.StationID
WHERE s.StationCode = 'MALTT' AND af.IsAvailable = 1;
```

All Rows Fetched: 2 in 0.01 seconds

	ACCESSIBILITYFEATUREID	TYPENAME	ISAVAILABLE	STATIONID	STATIONID_1	STATIONCODE	NAME	STREETADDRESS	CITY
1	1	Wheel Chair	1	102	102	MALTT	Malton Station	7878 Kirlin Cliff	Malton
2	3	Accessible washrooms	1	102	102	MALTT	Malton Station	7878 Kirlin Cliff	Malton

6.4.4 Fetch count of passenger having certain fare type

```
SELECT COUNT(p.Name) AS Passenger_Count_For_Fare_Type
FROM Passenger p
INNER JOIN FareType ft ON ft.FareTypeID = p.FareTypeID
WHERE ft.TypeName = ’Student Fare’
GROUP BY ft.TypeName;
```



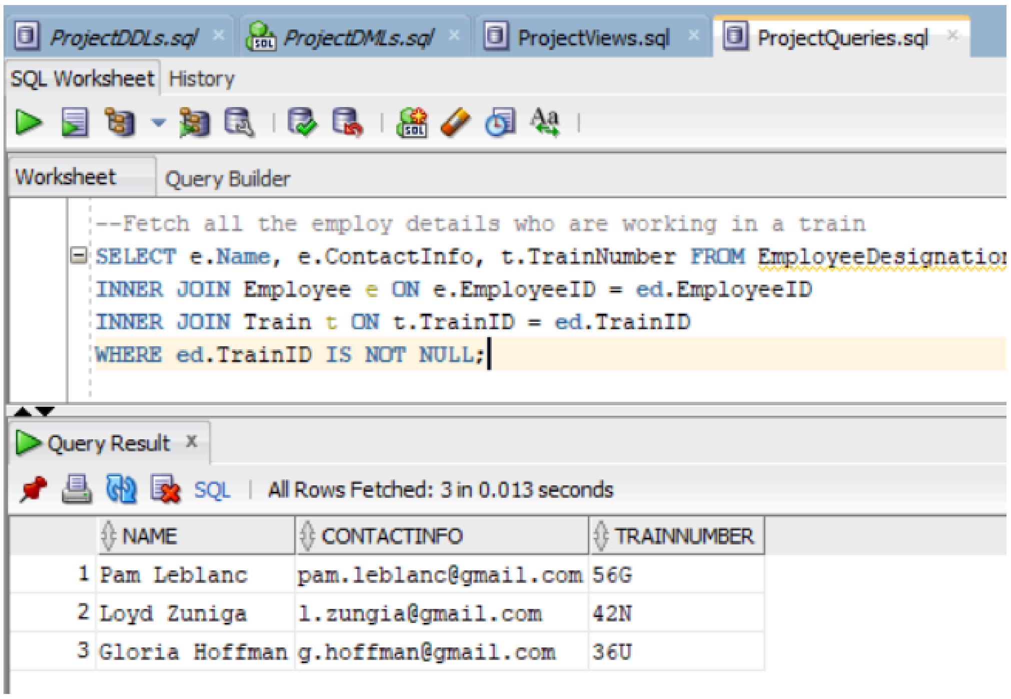
```
--Fetch count of passenger having certain fare type
SELECT COUNT(p.Name) AS Passenger_Count_For_Fare_Type
FROM Passenger p
INNER JOIN FareType ft ON ft.FareTypeID = p.FareTypeID
WHERE ft.TypeName = 'Student Fare'
GROUP BY ft.TypeName;
```

All Rows Fetched: 1 in 0.011 seconds

	PASSENGER_COUNT_FOR_FARE_TYPE
1	1

6.4.5 Fetch all the employ details who are working in a train

```
SELECT e.Name, e.ContactInfo, t.TrainNumber FROM EmployeeDesignation ed
INNER JOIN Employee e ON e.EmployeeID = ed.EmployeeID
INNER JOIN Train t ON t.TrainID = ed.TrainID
WHERE ed.TrainID IS NOT NULL;
```



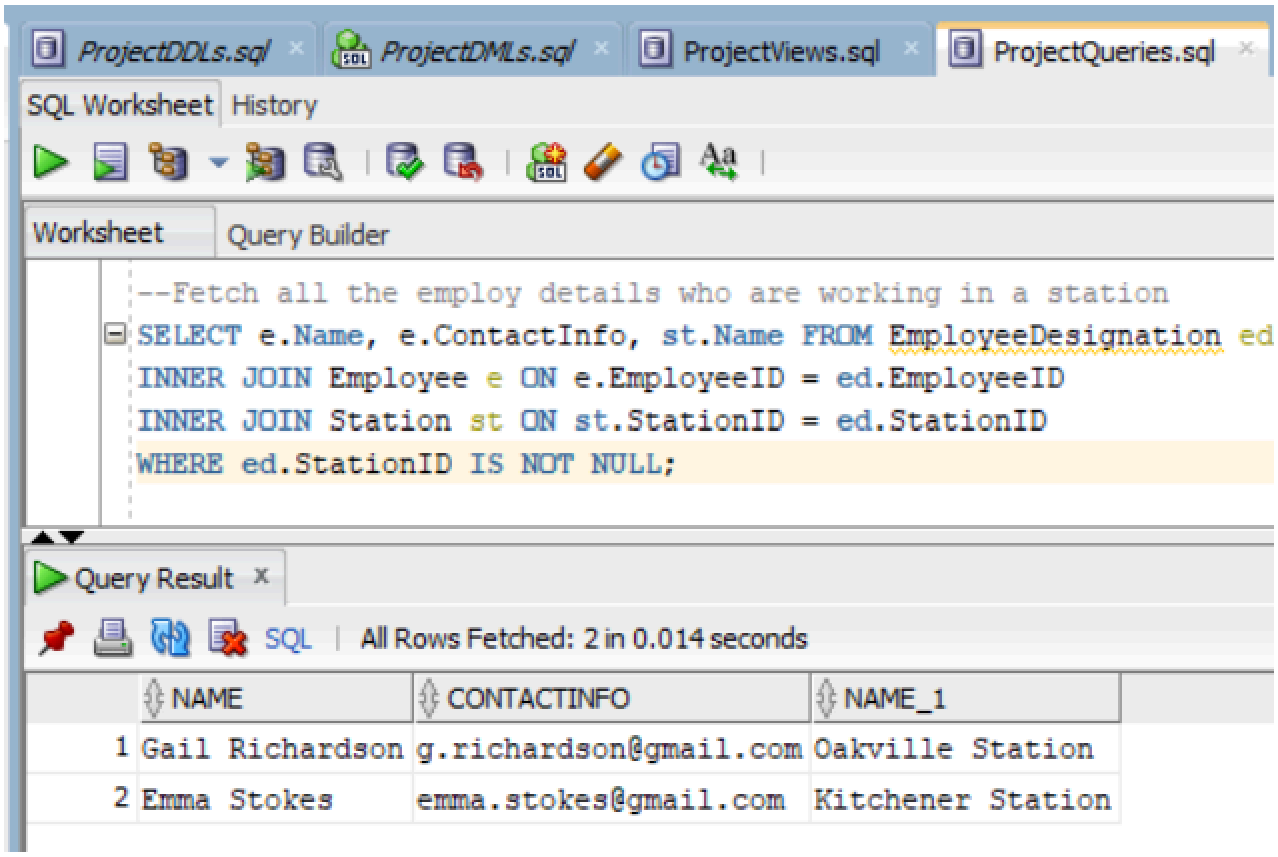
```
--Fetch all the employ details who are working in a train
SELECT e.Name, e.ContactInfo, t.TrainNumber FROM EmployeeDesignatio
INNER JOIN Employee e ON e.EmployeeID = ed.EmployeeID
INNER JOIN Train t ON t.TrainID = ed.TrainID
WHERE ed.TrainID IS NOT NULL;
```

All Rows Fetched: 3 in 0.013 seconds

	NAME	CONTACTINFO	TRAINNUMBER
1	Pam Leblanc	pam.leblanc@gmail.com	56G
2	Loyd Zuniga	l.zungia@gmail.com	42N
3	Gloria Hoffman	g.hoffman@gmail.com	36U

6.4.6 Fetch all the employ details who are working in a station

```
SELECT e.Name, e.ContactInfo, st.Name FROM EmployeeDesignation ed
INNER JOIN Employee e ON e.EmployeeID = ed.EmployeeID
INNER JOIN Station st ON st.StationID = ed.StationID
WHERE ed.StationID IS NOT NULL;
```



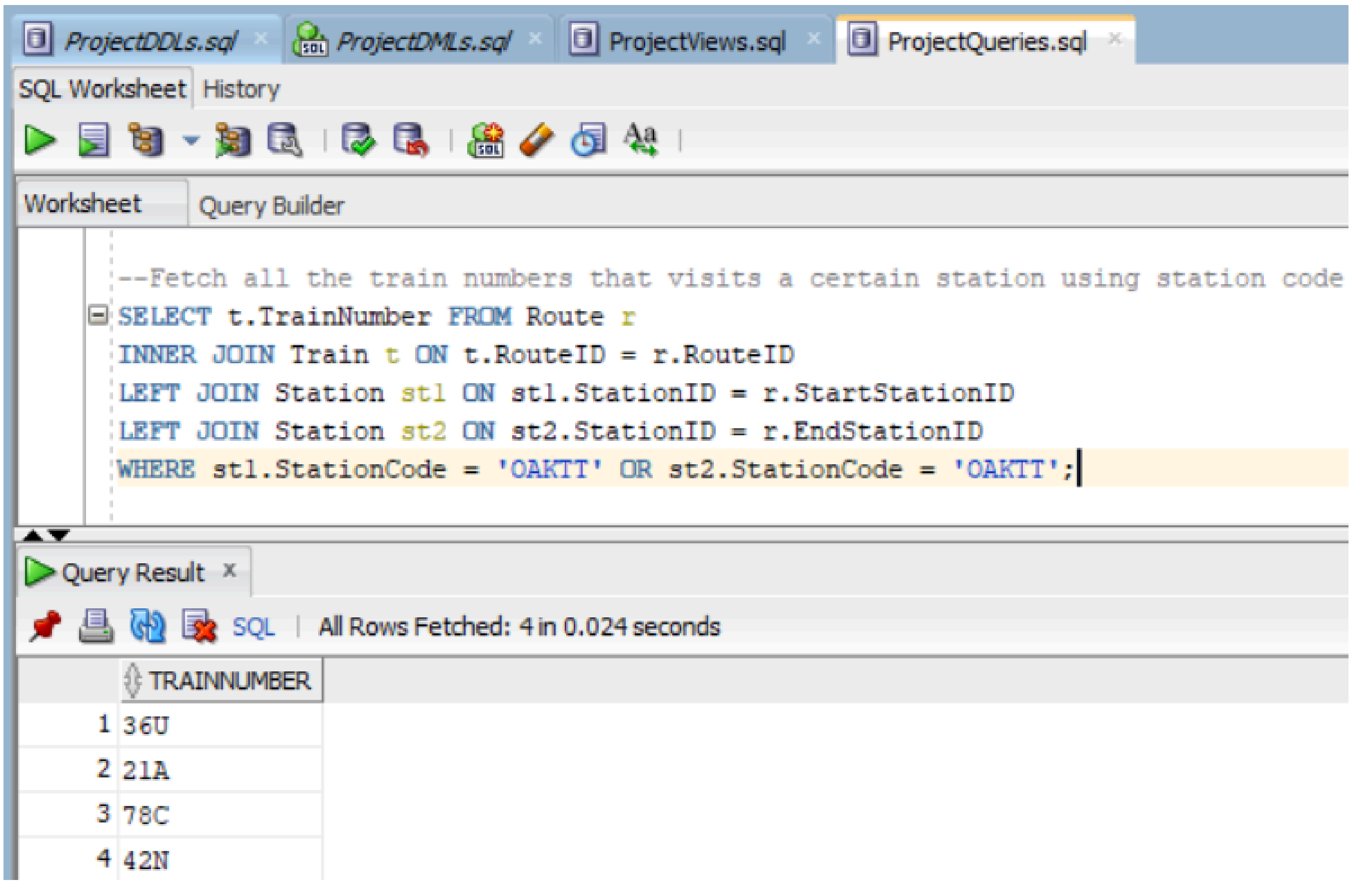
```
--Fetch all the employ details who are working in a station
SELECT e.Name, e.ContactInfo, st.Name FROM EmployeeDesignation ed
INNER JOIN Employee e ON e.EmployeeID = ed.EmployeeID
INNER JOIN Station st ON st.StationID = ed.StationID
WHERE ed.StationID IS NOT NULL;
```

All Rows Fetched: 2 in 0.014 seconds

	NAME	CONTACTINFO	NAME_1
1	Gail Richardson	g.richardson@gmail.com	Oakville Station
2	Emma Stokes	emma.stokes@gmail.com	Kitchener Station

6.4.7 Fetch all the train numbers that visits a certain station using station code

```
SELECT t.TrainNumber FROM Route r
INNER JOIN Train t ON t.RouteID = r.RouteID
LEFT JOIN Station st1 ON st1.StationID = r.StartStationID
LEFT JOIN Station st2 ON st2.StationID = r.EndStationID
WHERE st1.StationCode = 'OAKTT' OR st2.StationCode = 'OAKTT';
```



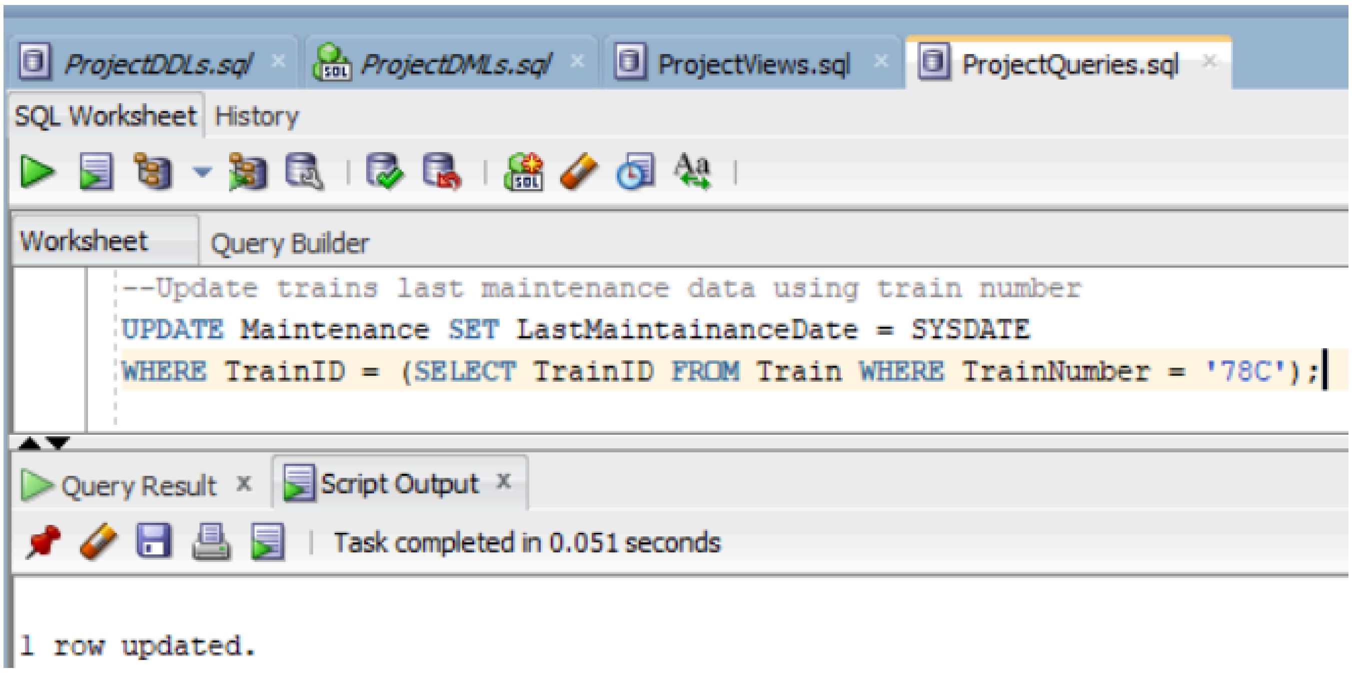
```
--Fetch all the train numbers that visits a certain station using station code
SELECT t.TrainNumber FROM Route r
INNER JOIN Train t ON t.RouteID = r.RouteID
LEFT JOIN Station st1 ON st1.StationID = r.StartStationID
LEFT JOIN Station st2 ON st2.StationID = r.EndStationID
WHERE st1.StationCode = 'OAKTT' OR st2.StationCode = 'OAKTT';
```

All Rows Fetched: 4 in 0.024 seconds

```
  TRAINNUMBER
1 36U
2 21A
3 78C
4 42N
```

6.4.8 Update trains last maintenance data using train number

```
UPDATE Maintenance SET LastMaintainanceDate = SYSDATE
WHERE TrainID = (SELECT TrainID FROM Train WHERE TrainNumber = '78C');
```



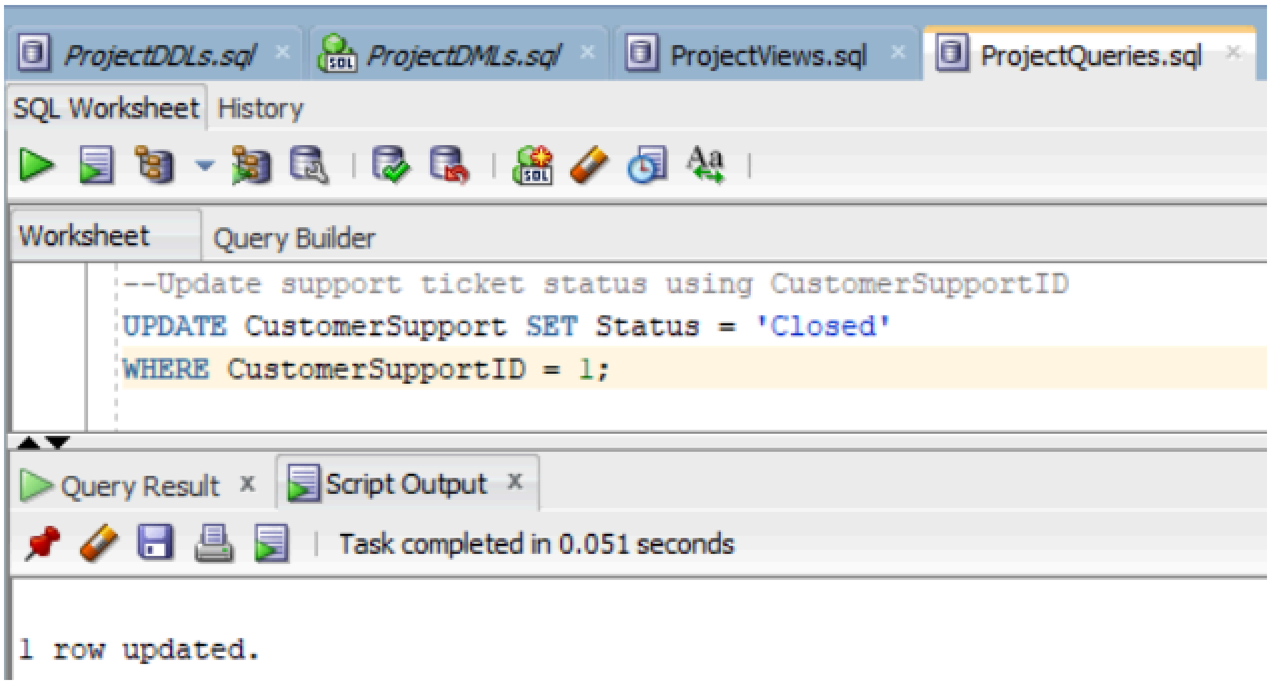
```
--Update trains last maintenance data using train number
UPDATE Maintenance SET LastMaintainanceDate = SYSDATE
WHERE TrainID = (SELECT TrainID FROM Train WHERE TrainNumber = '78C');
```

Task completed in 0.051 seconds

```
1 row updated.
```

6.4.9 Update support ticket status using CustomerSupportID

```
UPDATE CustomerSupport SET Status = 'Closed'
WHERE CustomerSupportID = 1;
```



```
--Update support ticket status using CustomerSupportID
UPDATE CustomerSupport SET Status = 'Closed'
WHERE CustomerSupportID = 1;
```

Task completed in 0.051 seconds

```
1 row updated.
```

7. CHALLENGES

7.1 Complexity of Business Rules: The Transit System Database needs to accommodate a wide range of complex business rules governing ticketing, scheduling, employee management, and maintenance. Implementing these rules accurately and efficiently within the database schema and application logic requires thorough understanding, careful interpretation, and precise implementation.

7.2 Real-Time Data Processing: The transit system operates in real-time, requiring the database to handle a continuous stream of data updates, transactions, and events from various sources such as ticketing machines, GPS devices, and maintenance logs. Implementing efficient data processing pipelines, event-driven architectures, and transaction management mechanisms is essential to ensure timely and accurate data processing.

7.3 Security and Compliance: Ensuring the security and compliance of the database with regulations such as GDPR, HIPAA, and industry standards for data protection is paramount. Implementing robust authentication mechanisms, access controls, encryption techniques, and auditing procedures to safeguard sensitive data and ensure compliance with regulatory requirements is essential.

8. CONCLUSION

In summary, the development of the Transit System Database enhances the current transit management by adoption of efficient database system principles. It presented a concerted effort to redefine transit operations by harnessing the power of effective data management. Throughout its journey, the project navigated through critical phases such as requirement gathering, design and modelling, and database scripting. Stakeholder collaboration played a pivotal role in shaping the project's direction, with robust tools and techniques employed to ensure comprehensive data collection and analysis. From conceptualizing the database structure to implementing it into a physical schema, each step was meticulously executed to align with the goal of optimizing transit systems.

Central to the project's success were the development of major design elements, including entities, tables, and relationships, depicted through the Entity Relationship Diagram. Challenges encountered along the way, such as business rules complexities and compliance requirements, underscored the importance of adaptive problem-solving and collaborative teamwork. Despite these hurdles, the team committed to revolutionizing transit operations, paving the way for a more efficient, connected, and sustainable public transportation ecosystem. As the Transittrack Database Project nears its end, its impact on transit management is set to be profound, setting a new standard for data-driven innovation in the transportation sector.

9. REFERENCES

OFFICIAL DOCUMENTATION AND STANDARDS

Oracle. (2023). Oracle Database Documentation.
[https://www.oracle.com/database/technologies/]

IEEE. (2023). IEEE Guide for Developing System Requirements Specifications.
[https://standards.ieee.org/]

BOOKS AND ACADEMIC JOURNALS

Coronel, C., Morris, S., & Rob, P. (2017). *Database Systems Design, Implementations and Management* (13th ed.)
Ramakrishnan, R., & Gehrke, J. (2002). *Database Management Systems*.
Hoffer, J. A., Ramesh, V., & Topi, H. (March 9, 2022). *Modern Database Management*. Published by Pearson.

ONLINE RESOURCES

Email validation check with Oracle regular expressions, *Oracle PL/SQL tips by Boobal Ganesan*
https://www.dba-oracle.com/t_email_validation_regular_expressions.htm

W3Schools. (2023). SQL Tutorial.

[https://www.w3schools.com/sql/]

TechOnTheNet. (2023). Oracle / PLSQL: Foreign Keys.
[https://www.techonthenet.com/oracle/foreign_keys.php]

TECHNICAL FORUMS AND COMMUNITIES

Stack Overflow. (2023). Database Design Questions and Answers.
[https://stackoverflow.com/]

Database Administrators Stack Exchange. (2023). Questions Tagged with 'Oracle'.
[https://dba.stackexchange.com/questions/tagged/oracle]

10. APPENDIX

- Team Chatter – pdf document containing detailed information on the team
- Database Topic and Mission Statement - pdf containing background of the project
- Database Design and Initial ERD - pdf containing the database design
- ProjectDDLs.sql - script containing all the statements for table creation and alter statements to add constraints
- ProjectDMLs.sql - script containing all the insert statements created using Oracle SQL Developer
- ProjectQueries.sql - script containing miscellaneous queries for the database created using Oracle SQL Developer
- ProjectViews.sql - script containing all view creation statements created using Oracle SQL Developer
- Readme – pdf containing step by step guide for running all scripts