Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

Train Scheduling System Database

Eligijus Skersonas

19335661

skersone@tcd.ie

Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

Contents

Section A

Application Design

The Train Scheduling Database System is a system that can be used by a Railway company to schedule their trains and provide information to their passengers such as departure and arrival times of different trains from various different stations. The system provides the basics of a schedule system for a railway company. For security and protective measures the database system has specific roles and permissions which help prevent the corruption of data. There are also triggers and constraints to ensure that the data stored is clean and what we would expect.

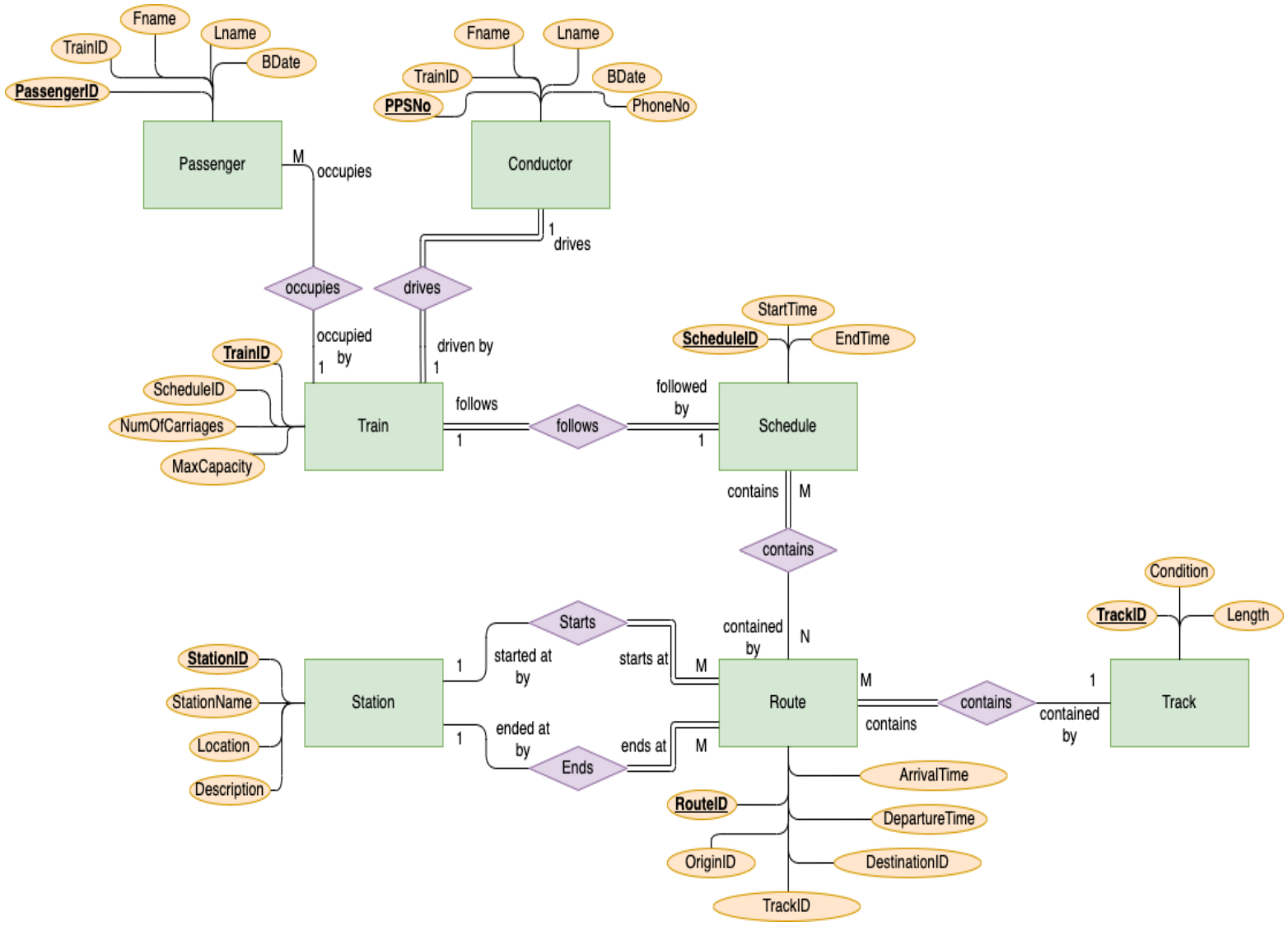
Assumptions made:

- In total there are 7 entities in the database which are Passenger, Conductor, Train, Schedule, Route, Station, Track
- Passenger Entity
  - Has 5 attributes which are PPSNo, TrainID, Fname, Lname, BDate
  - Every Passenger has already booked a ticket
  - Each Passenger CAN occupy 1 Train at a given time
- Conductor Entity
  - Has 6 attributes which are PPSNo, TrainID, Fname, Lname, BDate, PhoneNo
  - Every Conductor has a licence and is employed by the Train company
  - Each Conductor MUST drive 1 Train
- Train Entity
  - Has 4 attributes which are TrainID, ScheduleID, MaxCapacity, NoOfCarriages
  - Every Train is in working condition
  - Each Train MUST follow 1 Schedule
  - Each Train MUST be driven by 1 Conductor
  - Each Train CAN be occupied by many Passengers

- Schedule Entity
  - Has 3 attributes which are ScheduleID, StartTime, EndTime
  - Each Schedule MUST contain many Routes
  - Each Schedule MUST be followed by 1 Train
  - StartTime and EndTime indicate the Train journeys start and end time
- Route Entity
  - Has 6 attributes which are RouteID, OriginID, TrackID, DestinationID, DepartureTime, ArrivalTime
  - Each Route CAN be contained in many Schedules
  - Each Route MUST start in 1 Station
  - Each Route MUST end in 1 Station
  - Each Route MUST contain 1 Track
  - Start and End DOES NOT indicate that the Train Starts/Ends its journey at the Station in question. Start and End are rather used to indicate the direction of the train (stations acting as nodes, and the track as a directed edge)
  - Departure and arrival times indicate when the Train will depart from OriginID and arrive at DestinationID
  - The Train will stop at OriginID and DestinationID to pick up passengers

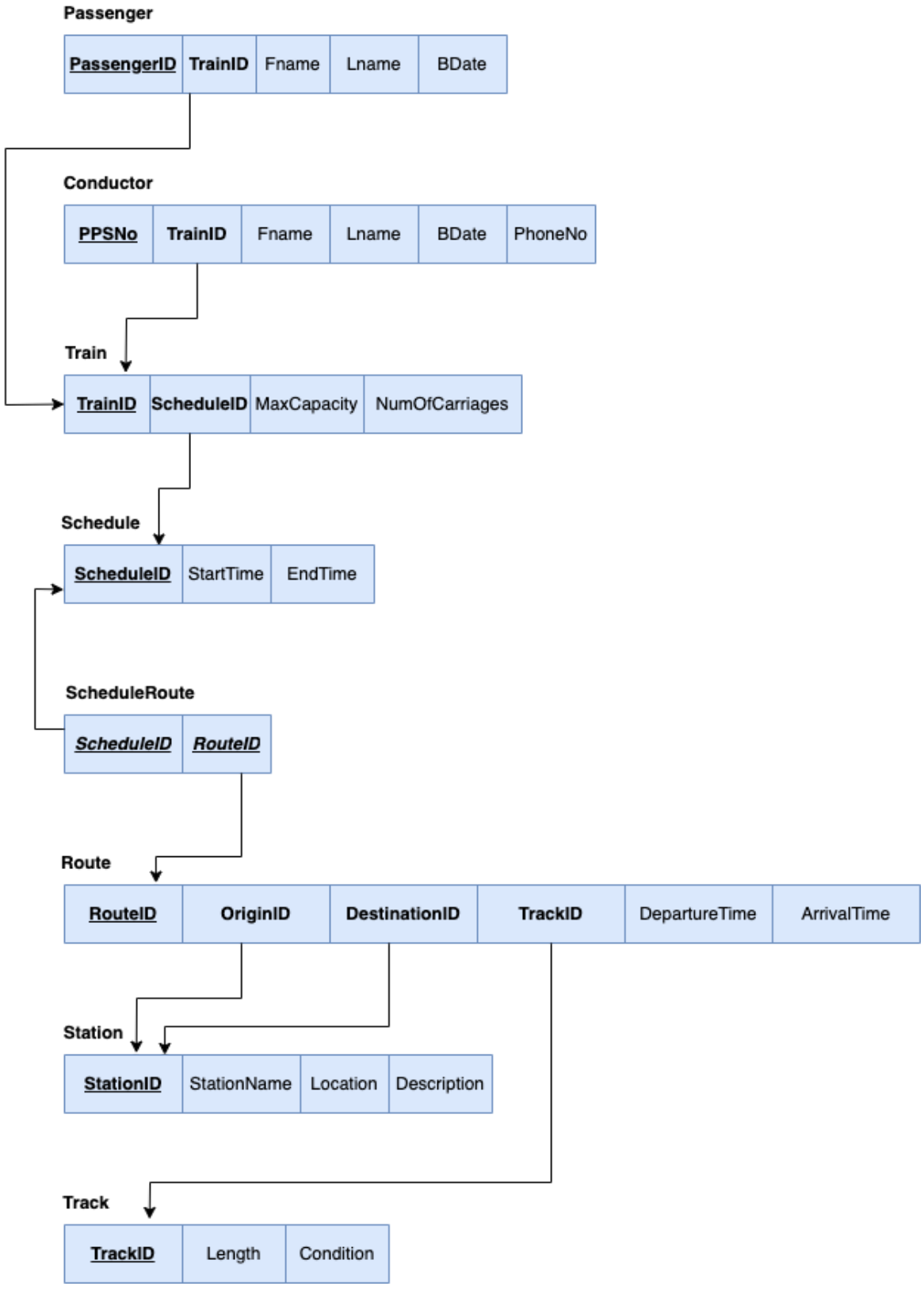
- Station Entity
  - Has 4 attributes which are StationID, StationName, Location, Description
  - Each Station CAN be started at by many Routes
  - Each Station CAN be ended at by many Routes
- Track Entity
  - Has 3 attributes which are TrackID, Condition, Length
  - Each Track CAN be contained in many Routes
  - Not every track is in use yet, as some may be under repairs
  - Condition will indicate whether the track is available to trains or not

Entity Relationship Diagram



Mapping to Relational Schema

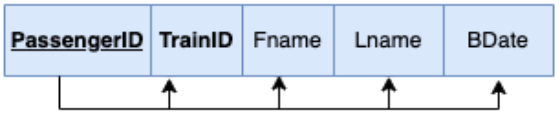
Primary keys are of the form **PrimaryKey**
Foreign Keys are of the form **Foreign Key**
If both a primary and foreign key then it is of the form ***Key***



Functional Dependency Diagrams

Primary keys are of the form **PrimaryKey**
Foreign Keys are of the form **Foreign Key**
If both a primary and foreign key then it is of the form ***Key***

Passenger



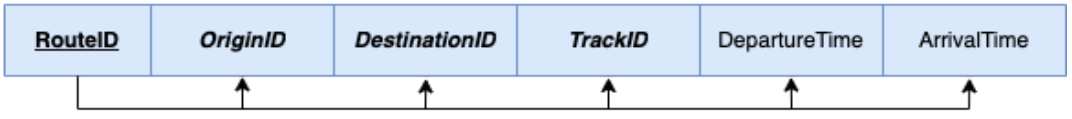
Conductor

Train

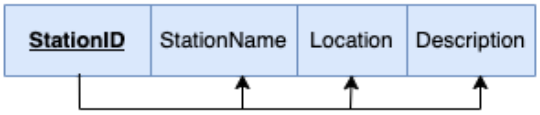
Schedule

ScheduleRoute

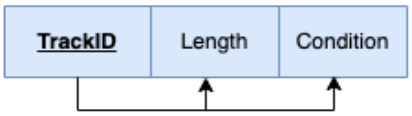
Route



Station



Track



Section B

Explanation of one of the SQL Codes for creating one of the database tables

```
CREATE TABLE `Conductor`(`PPSNo` VARCHAR(9) NOT NULL CHECK (`PPSNo` REGEXP
    '[1-9][0-9][0-9][0-9][0-9][0-9][0-9][A-Z][A-Z]?'),
    `TrainID` INTEGER NOT NULL,
    `Fname` VARCHAR(255) NOT NULL,
    `Lname` VARCHAR(255) NOT NULL,
    `BDate` DATE NOT NULL,
    `PhoneNo` VARCHAR(15) NOT NULL,
    PRIMARY KEY (`PPSNo`),
    FOREIGN KEY (`TrainID`) REFERENCES `Train` (`TrainID`)
);
```

In this section I will explain the SQL code for creating the **Conductor** database table. Inside the parentheses of the CREATE TABLE command the 6 attributes of the Conductor Entity are created in the table along with their data type and some constraints and after creating the attributes we then set any primary or foreign keys.

- **Data types**
  - **PPSNo** data type is set to VARCHAR(9) because a PPS number is 7 digits long followed by 1 or 2 letters.
  - **TrainID** data type is set to INTEGER because it is a foreign key to the Train table where TrainID is of type INTEGER
  - **Fname** and **Lname** data type is set to VARCHAR(255) to store any first and last names
  - **BDate** data type is set to DATE which will store their date of birth in the format yyyy-mm-dd
  - **PhoneNo** data type is set to VARCHAR(15) as this limit will include all the different lengths of mobile numbers
- **Constraints**
  - All attributes have a NOT NULL constraint is used to ensure they are not null
  - **PPSNo** has a CHECK constraint is used to check if the PPSNo is of PPS number format
- **Primary and Foreign Keys**
  - **PPSNo** is set as the Primary Key via the command PRIMARY KEY(`PPSNo`)
  - **TrainID** is set as a Foreign Key referencing the TrainID primary key in Train

Explanation and SQL Code for Altering Tables

```
ALTER TABLE `Conductor`
MODIFY COLUMN `PhoneNo` VARCHAR(20);
```

In this section I have provided SQL code for altering the tables. The code above alters the datatype of the PhoneNo attribute in Conductor to allow for larger phone number as some phone numbers exceed 15 digits/characters

Explanation and SQL Code for Trigger Operations

```
DROP TRIGGER IF EXISTS `PassengerTrigger`;

delimiter //
    CREATE TRIGGER `PassengerTrigger` BEFORE INSERT ON `Passenger`
    FOR EACH ROW BEGIN
      IF (NEW.`Fname` IS NULL) THEN
            SET NEW.`Fname` = 'Unknown';
      END IF;
      IF (NEW.`Lname` IS NULL) THEN
            SET NEW.`Lname` = 'Unknown';
      END IF;
    END; //
delimiter ;

DROP TRIGGER IF EXISTS `TrainTrigger`;

delimiter //
    CREATE TRIGGER `TrainTrigger` BEFORE INSERT ON `Train`
    FOR EACH ROW BEGIN
      IF (NEW.`MaxCapacity` < 0) THEN
            SET NEW.`MaxCapacity` = 0;
      END IF;
      IF (NEW.`NumOfCarriages` < 0) THEN
            SET NEW.`NumOfCarriages` = 0;
      END IF;
    END; //
delimiter ;
```

```
DROP TRIGGER IF EXISTS `TrackTrigger`;

delimiter //
    CREATE TRIGGER `TrackTrigger` BEFORE INSERT ON `Track`
    FOR EACH ROW BEGIN
      IF (NEW.`Length` < 0) THEN
            SET NEW.`Length` = 0;
      END IF;
        IF (NEW.`Condition` NOT IN ('available','unavailable')) THEN
            SET NEW.`Condition` = 'unavailable';
      END IF;
    END; //
delimiter ;
```

In this section I have provided SQL code of three Trigger Operations. For each of the following trigger operations:

- **PassengerTrigger** - this trigger changes any NULL name attributes into "unknown" before inserting into the Passenger Table
- **TrainTrigger** - this trigger changes the attributes MaxCapacity and NumOfCarriages to 0 before inserting into the Train Table if the value that is being inserted is less than 0. This helps prevent any errors that may be thrown by the constraints put on these attributes.
- **TrackTrigger** - this trigger changes the attribute Length before inserting into the Track table if the value provided is less than 0. This trigger also changes any string that was provided for the Condition attribute to "unavailable" if it does not match the constraint on the attribute. An example would be if someone provided the value "under repair" then it will change to "unavailable" before being inserted into the table.

Explanation and SQL Code for Creation of Views

```
CREATE VIEW `StartStations` AS
SELECT DISTINCT `StationID`,`StationName`
FROM `Station`, `Route`
WHERE `StationID` = `Route`.`OriginID`;

CREATE VIEW `EndStations` AS
SELECT DISTINCT `StationID`,`StationName`
FROM `Station`, `Route`
WHERE `StationID` = `Route`.`DestinationID`;

CREATE VIEW `TrainSchedules` AS
SELECT `Train`.`TrainID`, `Schedule`.`StartTime` AS `ScheduleStartTime`,
            `Schedule`.`EndTime` AS `ScheduleEndTime`,
            `StartStations`.`StationName` AS `FromStation`,
`Route`.`DepartureTime`,
        `EndStations`.`StationName` AS `ToStation`,
`Route`.`ArrivalTime`
FROM `Train`, `Schedule`, `StartStations`, `EndStations`, `Route`,
`ScheduleRoute`
WHERE `Train`.`ScheduleID` = `Schedule`.`ScheduleID` AND
            `Schedule`.`ScheduleID` = `ScheduleRoute`.`ScheduleID` AND
        `ScheduleRoute`.`RouteID` = `Route`.`RouteID` AND
        `Route`.`OriginID` = `StartStations`.`StationID` AND
        `Route`.`DestinationID` = `EndStations`.`StationID`
ORDER BY `TrainID` ASC, `DepartureTime` ASC;
```

In this section I've provided SQL code for creating views. The first two views created are **EndStations** and **StartStation** which will provide tables for stations where trains depart from and stations from where trains arrive at.
The view **TrainSchedule** makes use of the other two views to make a view of all the train schedules where we are able to see the time a trains schedule starts and ends, the stations that the train departs and arrives at and their corresponding departure and arrival times. I have also changed some of the column names for this view so that we avoid the error of having two columns with the same name and also to increase readability/understandability of the table. **TrainSchedule** is also ordered by ascending TrainID and ascending DepartureTime.

Explanation and SQL Code for populating one of the tables

```
INSERT INTO `Conductor` (`PPSNo`,`TrainID`,`Fname`,`Lname`,`BDate`,`PhoneNo`)
VALUES
    ('1234567AB', 1, 'John', 'Doe', '2000-10-02', '0831231234'),
    ('7654321BA', 2, 'Tom', 'Cruise', '1980-11-02', '0835432121'),
    ('1111222A', 3, 'Bob', 'Hamilton', '1990-07-23', '0851212984'),
    ('3213213P', 4, 'Tom', 'Hiddleston', '2000-08-20', '0867634512'),
    ('2132435Z', 5, 'Tom', 'Jackson', '1999-01-09', '0879236745');
```

In this section I have provided the SQL which will populate the conductor table as an example. For the **Conductor** table I inserted 5 different Conductors. Each conductor has its own unique PPSNo which is a primary key and follows the constraint for the PPSNo attribute. Each driver also has a unique train that they drive. I have also populated random names, phone numbers and also dates of birth which are of yyyy-mm-dd format. The data type that was used for BDate will allow us to sort by date of birth if desired.

Explanation and example of SQL code retrieving information

```
SELECT *
FROM `TrainSchedules`
WHERE `TrainSchedules`.`TrainID` = 1
ORDER BY `TrainID` ASC, `DepartureTime` ASC;

SELECT *
FROM `TrainSchedules`
WHERE `TrainSchedules`.`TrainID` = 2
ORDER BY `TrainID` ASC, `DepartureTime` ASC;

SELECT *
FROM `TrainSchedules`
WHERE `TrainSchedules`.`TrainID` = 3
ORDER BY `TrainID` ASC, `DepartureTime` ASC;

SELECT *
FROM `TrainSchedules`
WHERE `TrainSchedules`.`TrainID` = 4
ORDER BY `TrainID` ASC, `DepartureTime` ASC;

SELECT *
FROM `TrainSchedules`
WHERE `TrainSchedules`.`TrainID` = 5
ORDER BY `TrainID` ASC, `DepartureTime` ASC;
```

Schedule obtained from the first select query where TrainID = 1

TrainID	ScheduleStartTi...	ScheduleEndTime	FromStation	DepartureTime	ToStation	ArrivalTime
1	12:00:00	12:30:00	Heuston Station	12:00:00	Connolly Station	12:12:00
1	12:00:00	12:30:00	Connolly Station	12:12:00	Tara Street Station	12:25:00
1	12:00:00	12:30:00	Tara Street Station	12:25:00	Pearse Street Station	12:30:00

Schedule obtained from the first select query where TrainID = 2

TrainID	ScheduleStartTi...	ScheduleEndTime	FromStation	DepartureTime	ToStation	ArrivalTime
2	09:00:00	09:30:00	Heuston Station	09:00:00	Greystones Station	09:30:00

Schedule obtained from the first select query where TrainID = 3

TrainID	ScheduleStartTi...	ScheduleEndTime	FromStation	DepartureTime	ToStation	ArrivalTime
3	12:30:00	13:00:00	Pearse Street Station	12:30:00	Tara Street Station	12:35:00
3	12:30:00	13:00:00	Tara Street Station	12:35:00	Connolly Station	12:48:00
3	12:30:00	13:00:00	Connolly Station	12:48:00	Heuston Station	13:00:00

Schedule obtained from the first select query where TrainID = 4

TrainID	ScheduleStartTi...	ScheduleEndTime	FromStation	DepartureTime	ToStation	ArrivalTime
4	20:30:00	21:00:00	Pearse Street Station	20:30:00	Heuston Station	21:00:00

Schedule obtained from the first select query where TrainID = 5

TrainID	ScheduleStartTi...	ScheduleEndTime	FromStation	DepartureTime	ToStation	ArrivalTime
5	16:30:00	16:50:00	Pearse Street Station	16:30:00	Connolly Station	16:50:00

The select queries above make use of the **TrainSchedules** view to produce the schedules shown above. Each of the queries select all the tuples where the TrainID matches the ID constant given in the WHERE clause. Each schedule is ordered by ascending TrainID and ascending DepartureTime.

Explanation and SQL Code of Security and Permissions

I have used security for this database to avoid any corruption of data. I have created and set some roles with specific privileges granted for each role. For this database the roles I have gone with are SystemAdmin, Conductor and Passenger.

```
CREATE ROLE 'sys_admin', 'conductor', 'passenger';

GRANT SELECT ON `trainScheduleSystem`.`TrainSchedules` TO 'passenger',
'conductor';
GRANT SELECT ON `trainScheduleSystem`.`Train` TO 'conductor';
GRANT ALL ON `trainScheduleSystem` TO 'sys_admin';

CREATE USER 'sys_admin1' IDENTIFIED BY '1000';
CREATE USER 'sys_admin2' IDENTIFIED BY '1001';
CREATE USER 'conductor1' IDENTIFIED BY '1002';
CREATE USER 'passenger1' IDENTIFIED BY '1003';

GRANT 'passenger' to 'passenger1';
GRANT 'conductor' to 'conductor1';
GRANT 'sys_admin' to 'sys_admin1';
GRANT 'sys_admin' to 'sys_admin2';

REVOKE ALL PRIVILEGES, GRANT OPTION FROM 'sys_admin';
```

The SQL code provided above creates three roles which are **sys_admin**, **conductor** and **passenger**. Then I have granted certain privileges for the different roles such as the select action for **passenger** and **conductor** on the TrainSchedules view. Then I create 4 users and grant each user a role. If we wish to, we can also revoke all privileges from a role using the revoke SQL code above.

Section C

(Note Section C should be submitted as a separate file which is a listing of your SQL script from MySQL)