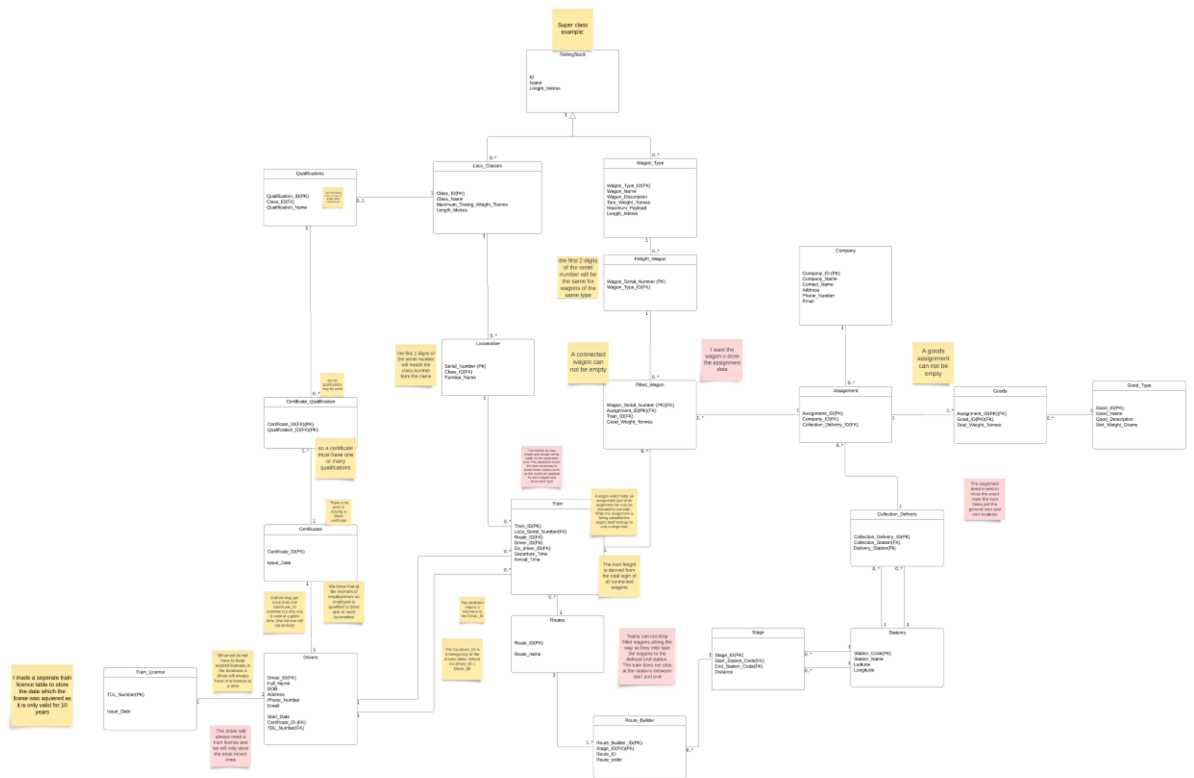
Introduction

The database system was designed to cater to the specific needs of Swift Southern Railway (SSR), enabling the efficient allocation of rolling stock to trains to develop schedules.

This report aims to outline the decisions I made, from the conceptual design of the database to its implementation and testing with example queries. I will clarify the assumptions I made when making the data model, along with a discussion about why I chose to design the system as it is. This report aims to provide sufficient information to enable others to build upon the system in the future.

The Conceptual Data Model

Class Diagram



If, for whatever reason, this image is blurred, please look at the PDF version attached to my submission or use this link https://lucid.app/documents/embedded/9489897b-dc0b-4f06-8bfb-c0c7571441ee

Constraints and Assumptions

1. The system will only store the most up-to-date certificate for the driver. This is also true for the train licence. Furthermore, a driver must always have a valid certificate and train licence at any given time
2. An employed driver is already qualified to drive one or more locomotive classes. Thus, a certificate will be issued immediately and will not be blank.
3. Every driver can use any wagon
4. I'm assuming that an application will ensure an assignment's start and end locations match the train they are on. For example, suppose an assignment is collected from Manchester and delivered to London. In that case, the system should automatically assign the filled wagon to a train from Manchester to London.
5. The checks for the train's length, the locomotive's maximum towing weight, and the maximum payload for a wagon will be done by an application. This database stores the data necessary to make these checks, such as the length of each wagon and the total weight of goods stored on each wagon. Thus, the application will use this to allocate wagons and locomotives to carry out a good consignment.
6. A goods assignment can never be empty, as there would be no point in making/ accepting an order with nothing in it.
7. A Filled Wagon must have an assignment allocated to it, as there is no point in connecting an empty wagon to the train
8. The stock level of locomotives and wagons is reset daily, and the rolling stock's location does not matter. For this assignment, it is assumed that everything is available from anywhere (This is important to note when looking at the sample data and queries)
9. An empty train is generated first with a journey set. This makes a trainID available. The filled wagons which match the journey start and end stations will then use this trainID to get added to the train.
10. A train can carry assignments from more than one company at a time. So, it may be towing two filled wagons, each with an assignment from a different company.
11. A filled wagon cannot hold two assignments, only one part or whole of an assignment.

Design Decisions

Storing Totals

I decided not to store the values for any field which needs to be calculated, such as the total length of each train and the total gross weight of the connected wagons. Before discussing the benefits and implications of this, it is worth noting the differences between dynamic and static data. Dynamic data is data that changes frequently, and static data is data that stays consistent. Concerning this project, the fields that require calculation are dynamic, and data such as the length of a wagon or its maximum payload weight are static because these will most likely never change. The database stores the static information needed for the dynamic fields to be calculated at runtime and displayed in an SQL query view. I decided to design the database like this after considering the advantages and disadvantages of performance, integrity, and simplicity between storing total values or summing at runtime.

To explain the runtime performance advantage, I will use a train's "total gross weight" calculation as an example. This calculation is an aggregation of many wagons being summed up. Furthermore, the calculation to find the gross weight of a single wagon requires the addition of totals from two separate tables (the goods table and the freight wagon). The "total gross weight" is the sum of many parts. However, every part is a sum of other totals. Wagons can be added and removed from a train. Thus, If I stored the "total gross weight" as a field and changed a wagon's allocation, the system would have to recalculate every wagon's weight again, then sum them to finally give me the "total gross weight" of a train. The fact that wagons can be added and removed makes the calculation dynamic. That is crucial in deciding whether there would be a performance benefit in calculating at runtime or storing it. If the calculation were static and only calculated once and never changed, then holding the value would be best; however, due to many of the calculated fields in this project being dynamic, it is somewhat more performant to calculate them once at runtime. This introduces some problems, such as if data is manually entered into the database, there will not be any checks for the entered data to ensure that the train's max towing weight and length have not been exceeded. Instead, the checks to ensure that the maximum length of the train, the maximum towing weight of a locomotive, and the maximum payload for a wagon must be done by the application side only.

Calculating the total at runtime ensures consistency between the total value and the actual data, eliminating the need to update or synchronise the total every time the underlying data changes. So, if I only store the parts that make up the calculation, such as the weight and lengths of a filled wagon, and then calculate the result train gross weight in a query at run time, I will have the correct calculation, even if I make changes to the wagons.

Lastly, making an SQL query that calculates everything at the end is much easier than setting up triggers, which sum up as the train is being set up. Sometimes, a calculation must reference a column in the same row. This could occur when the calculation checks the wagon type to determine the length or maximum weight that should be allowed. This referencing of the same column can cause many issues. Firstly, the row would have to be made first with the fields which need to be calculated set to null, then the triggers can reference the same row. If this wasn't done, the calculations would have to be done at the same time as the object/row is created which is not possible.

Feedback changes

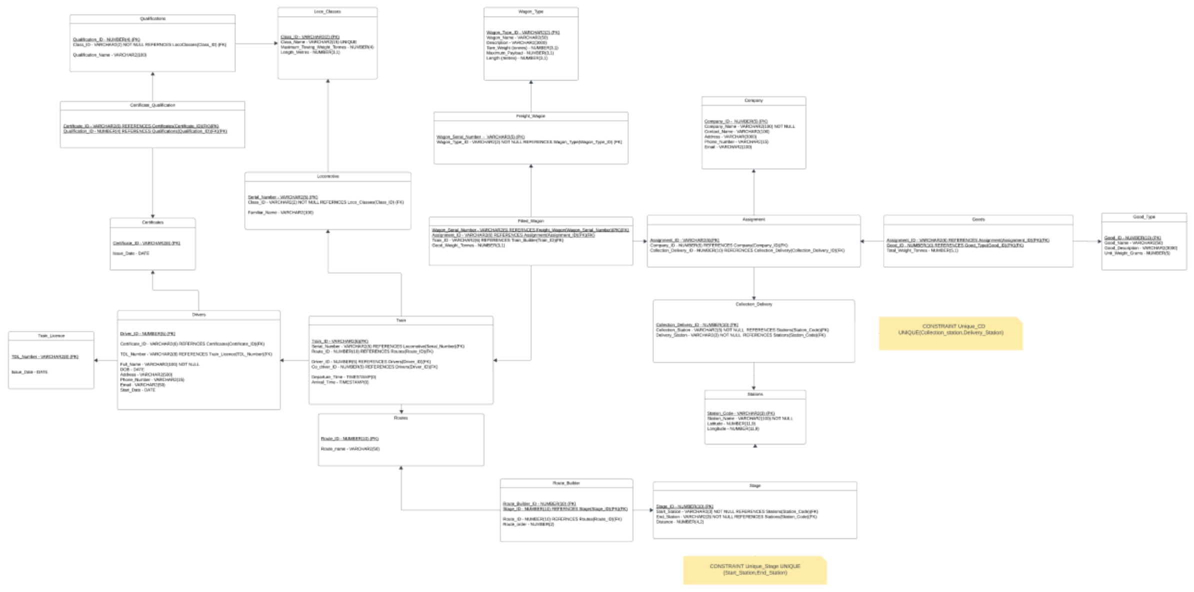
The feedback I received for my initial class diagram was, "How do you identify which wagons are being allocated to a train. Consider using supertype/subtypes". The first issue arrows because of how I had initially thought about the multiplicity between "Trains" and "Filled_Wagons". At first, the "Filled_Wagons" class was called "Connected_Wagons", which acted as a wagon accumulator. The multiplicity error occurred when I assumed that these set accumulated wagons could be allocated to many different trains and that one train could only have one set of connected wagons. However, this raises the issue of which set of wagons belonged to which train and what happens when a different combination of wagons is used. An entirely new connected wagon must be made and given an ID; thus, every combination of wagons would have to be accounted for, which would be a computational nightmare and borderline impossible as it would take an unfathomable amount of time to compute. To solve this problem, I reversed the multiplicity; thus, each wagon had a link to the train to which it would be allocated.

Superclass

The locomotives and freight wagon class shared the attributes of ID, Name and Length_Metres. I made a superclass called "Rolling_Stock", which had these attributes, and both the locomotive and freight wagon classes would then inherit these attributes (Note: I renamed them to represent better the class in which they were inherited). During implementation, I did not create a table called Rolling_Stock. Instead, as shown in the relation model, I duplicated the attributes into locomotive and freight wagon tables and renamed them accordingly. Implementing the superclass this way was done after considering the advantages and disadvantages of alternative options.

One such way of implementing the Rolling_Stock superclass is storing the common attributes in the superclass table and keeping the unique attributes in their respective locomotive or freight_wagon tables. The Rolling_Stock superclass will have two foreign keys referencing the locomotive and freight_wagon tables. This is where the issue of this implantation strategy becomes apparent: A rolling stock object can only be a locomotive or a freight wagon, not both. One of the foreign keys would have to be left null. This would make querying much harder as each row must be checked to ensure only one of the two keys has been entered. It would also have to check which key is not null as this determines whether it is a freight wagon or locomotive and, thus, what queries can be run on it. Furthermore, the primary goal of a foreign key is to maintain referential integrity. By allowing nullable foreign keys, there is a chance that the integrity of the data could become compromised, especially if both keys are left null. Thus, the Rolling_Stock object would not be a locomotive or a freight wagon. The only advantage of implementing the superclass this way would be that the locomotive and freight wagon classes would be kept smaller and potentially more manageable. While duplicating the attributes into both classes has made them larger, the advantages gained in query readability and improved referential integrity outweighed this slight disadvantage.

The Relational Data Model



If, for whatever reason, this image is blurred, please use this link https://lucid.app/documents/embedded/aaf204c8-d6fd-47a4-9c26-0d46dac59723

Implementation

```
CREATE TABLE Loco_Classes

(

   Class_ID VARCHAR2(2) PRIMARY KEY,

   Class_Name VARCHAR2(15) UNIQUE,

   Max_Towing_Weight_Tonnes NUMBER(4),

   Length_Metres NUMBER(3,1)

)


CREATE TABLE Locomotive

(

   Serial_Number VARCHAR2(5) PRIMARY KEY,

   Class_ID VARCHAR2(2) REFERENCES Loco_Classes(Class_ID),

   Familiar_Name VARCHAR2(100)

)
```

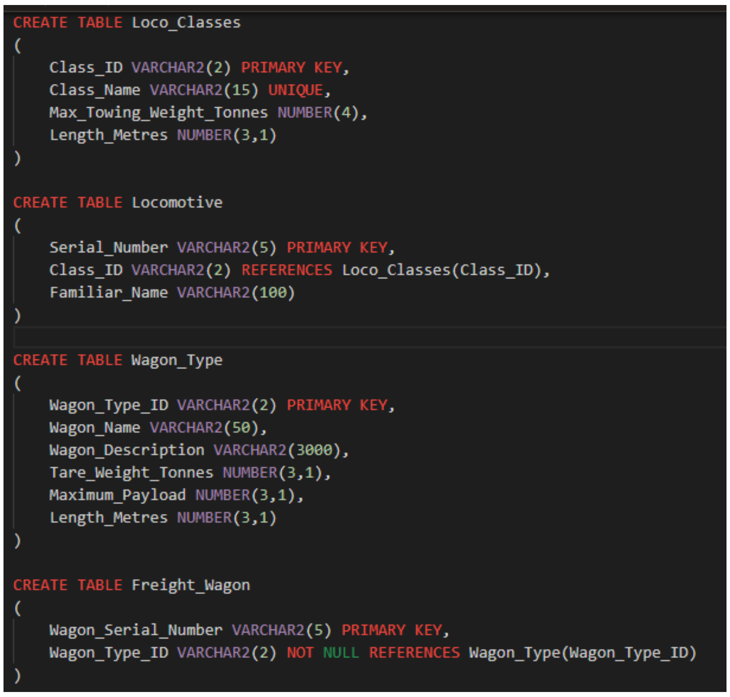
```
CREATE TABLE Wagon_Type
(
   Wagon_Type_ID VARCHAR2(2) PRIMARY KEY,
   Wagon_Name VARCHAR2(50),
   Wagon_Description VARCHAR2(3000),
   Tare_Weight_Tonnes NUMBER(3,1),
   Maximum_Payload NUMBER(3,1),
   Length_Metres NUMBER(3,1)
)

CREATE TABLE Freight_Wagon
(
   Wagon_Serial_Number VARCHAR2(5) PRIMARY KEY,
   Wagon_Type_ID VARCHAR2(2) NOT NULL REFERENCES Wagon_Type(Wagon_Type_ID)
)

CREATE TABLE Certificates
(
   Certificate_ID VARCHAR2(6) PRIMARY KEY,
   Issue_Date DATE
)

CREATE TABLE Qualifications
(
   Qualification_ID NUMBER(4) PRIMARY KEY,
   Class_ID VARCHAR2(2) NOT NULL REFERENCES Loco_Classes(Class_ID),
   Qualification_Name VARCHAR2(100)
)
```

```
CREATE TABLE Train_Licence
(
   TDL_Number VARCHAR2(8) PRIMARY KEY,
   Issue_Date DATE
)

CREATE TABLE Drivers
(
   Driver_ID NUMBER(5) PRIMARY KEY,
   Certificate_ID VARCHAR2(6) REFERENCES Certificates(Certificate_ID),
   TDL_Number VARCHAR2(8) REFERENCES Train_Licence(TDL_Number),
   Start_Date DATE,
   Full_Name VARCHAR2(100) NOT NULL,
   DOB DATE,
   Address VARCHAR2(500),
   Phone_Number VARCHAR2(15),
   Email VARCHAR2(50)
)

CREATE TABLE Stations
(
   Station_Code VARCHAR2(3) PRIMARY KEY,
   Station_Name VARCHAR2(100) NOT NULL,
   Latitude NUMBER(11,9),
   Longitude NUMBER(11,9)
)
```

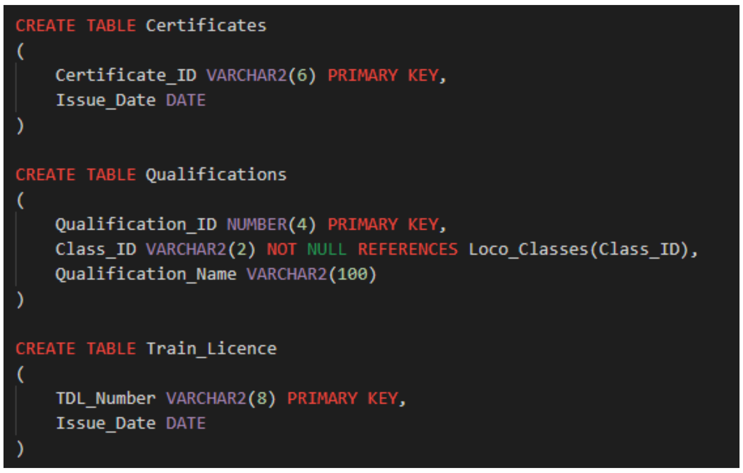
```
CREATE TABLE Stage
(
   Stage_ID NUMBER(10) PRIMARY KEY,
   Start_Station VARCHAR2(3) NOT NULL REFERENCES Stations(Station_Code),
   End_Station VARCHAR2(3) NOT NULL REFERENCES Stations(Station_Code),
   Distance NUMBER(5,2),
   CONSTRAINT Unique_Stage UNIQUE (Start_Station,End_Station)
)

CREATE TABLE Routes
(
   Route_ID NUMBER(10) PRIMARY KEY,
   Route_Name VARCHAR2(50)
)

CREATE TABLE Route_Builder
(
   Route_Builder_ID NUMBER(10),
   Stage_ID NUMBER(10) REFERENCES Stage(Stage_ID),
   Route_ID REFERENCES Routes(Route_ID),
   Route_Order NUMBER(2),
   PRIMARY KEY (Route_Builder_ID,Stage_ID)
)
```

```
CREATE TABLE Train
(
   Train_ID VARCHAR2(5) PRIMARY KEY,
   Loco_Serial_Number VARCHAR2(5) REFERENCES Locomotive(Serial_Number),
   Route_ID  NUMBER(10) REFERENCES Routes(Route_ID),
   Driver_ID NUMBER(5) REFERENCES Drivers(Driver_ID),
   Co_Driver_ID NUMBER(5) REFERENCES Drivers(Driver_ID),
   Departure_Time TIMESTAMP(0),
   Arrival_Time TIMESTAMP(0)
)

CREATE TABLE Company
(
   Company_ID NUMBER(5) PRIMARY KEY,
   Company_Name VARCHAR2(100) NOT NULL,
   Contact_Name VARCHAR2(100),
   Address VARCHAR2(3000),
   Phone_Number VARCHAR2(15),
   Email VARCHAR2(100)
)

CREATE TABLE Good_Type
(
   Good_ID NUMBER(10) PRIMARY KEY,
   Good_Name VARCHAR2(50),
   Good_Description VARCHAR2(3000),
   Unit_Weight_Grams NUMBER(5)
)
```

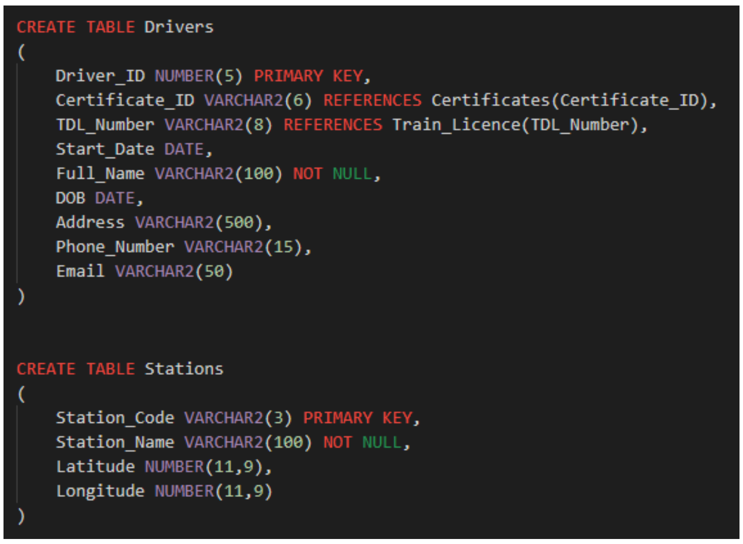
```
CREATE TABLE Collection_Delivery
(
   Collection_Delivery_ID NUMBER(10) PRIMARY KEY,
   Collection_station VARCHAR2(3) NOT NULL REFERENCES Stations(Station_Code),
   Delivery_Station VARCHAR2(3) NOT NULL REFERENCES Stations(Station_Code),
   CONSTRAINT Unique_CD UNIQUE(Collection_station,Delivery_Station)
)

CREATE TABLE Assignment
(
   Assignment_ID VARCHAR2(6) PRIMARY KEY,
   Company_ID NUMBER(5) REFERENCES Company(Company_ID),
   Collection_Delivery_ID NUMBER(10) REFERENCES Collection_Delivery(Collection_Delivery_ID)
)

CREATE TABLE Goods
(
   Assignment_ID VARCHAR2(6) REFERENCES Assignment(Assignment_ID),
   Good_ID NUMBER(10) REFERENCES Good_Type(Good_ID),
   Total_Weight_Tonnes NUMBER(5,1),
   PRIMARY KEY(Assignment_ID,Good_ID)
)

CREATE TABLE Filled_Wagon
(
   Wagon_Serial_Number  VARCHAR2(5) REFERENCES Freight_Wagon(Wagon_Serial_Number),
   Assignment_ID VARCHAR2(6) REFERENCES Assignment(Assignment_ID),
   Train_ID VARCHAR2(6) REFERENCES Train(Train_ID),
   Good_Weight_Tonnes NUMBER(3,1),
   PRIMARY KEY(Wagon_Serial_Number,Assignment_ID)
)
```



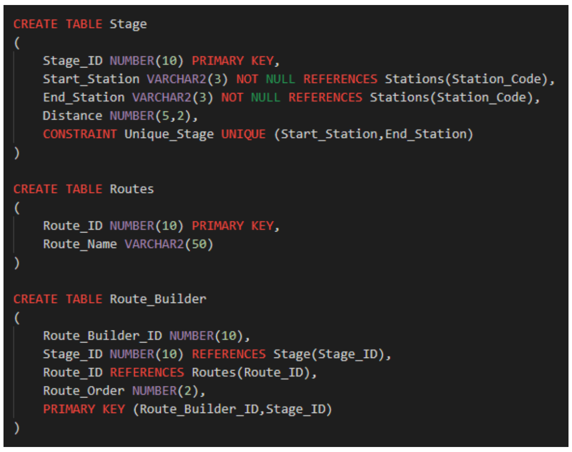
```sql
CREATE TABLE Loco_Classes
(
    Class_ID VARCHAR2(2) PRIMARY KEY,
    Class_Name VARCHAR2(15) UNIQUE,
    Max_Towing_Weight_Tonnes NUMBER(4),
    Length_Metres NUMBER(3,1)
)

CREATE TABLE Locomotive
(
    Serial_Number VARCHAR2(5) PRIMARY KEY,
    Class_ID VARCHAR2(2) REFERENCES Loco_Classes(Class_ID),
    Familiar_Name VARCHAR2(100)
)

CREATE TABLE Wagon_Type
(
    Wagon_Type_ID VARCHAR2(2) PRIMARY KEY,
    Wagon_Name VARCHAR2(50),
    Wagon_Description VARCHAR2(3000),
    Tare_Weight_Tonnes NUMBER(3,1),
    Maximum_Payload NUMBER(3,1),
    Length_Metres NUMBER(3,1)
)

CREATE TABLE Freight_Wagon
(
    Wagon_Serial_Number VARCHAR2(5) PRIMARY KEY,
    Wagon_Type_ID VARCHAR2(2) NOT NULL REFERENCES Wagon_Type(Wagon_Type_ID)
)
```



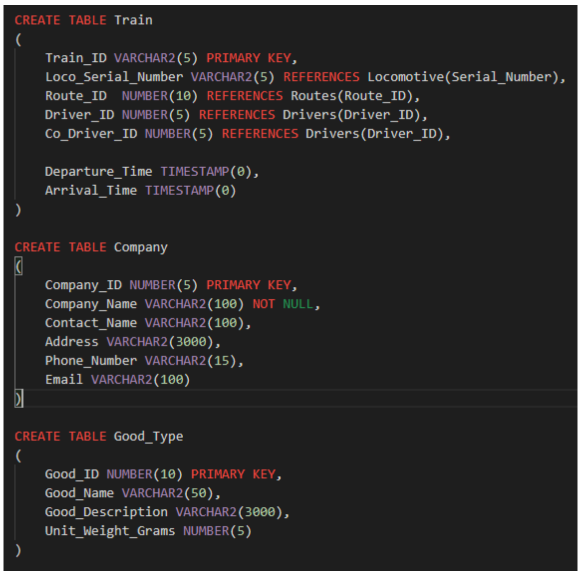
```sql
CREATE TABLE Certificates
(
    Certificate_ID VARCHAR2(6) PRIMARY KEY,
    Issue_Date DATE
)

CREATE TABLE Qualifications
(
    Qualification_ID NUMBER(4) PRIMARY KEY,
    Class_ID VARCHAR2(2) NOT NULL REFERENCES Loco_Classes(Class_ID),
    Qualification_Name VARCHAR2(100)
)

CREATE TABLE Train_Licence
(
    TDL_Number VARCHAR2(8) PRIMARY KEY,
    Issue_Date DATE
)
```



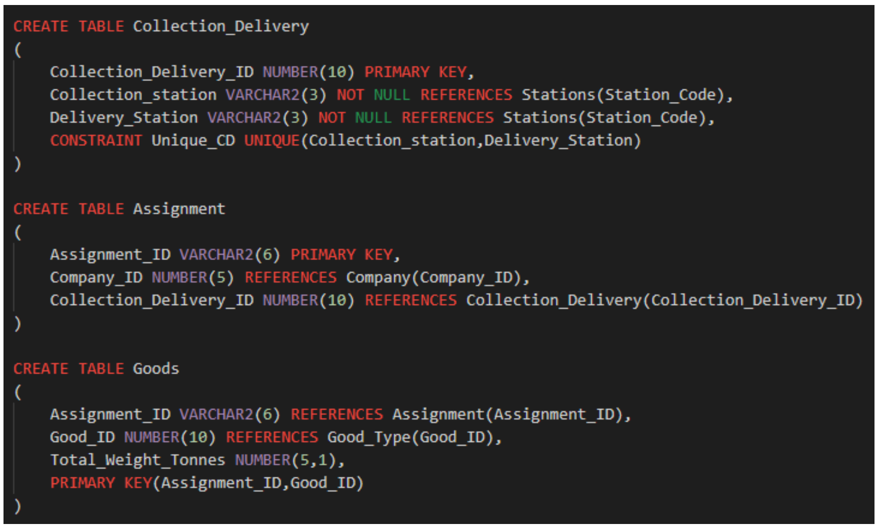
```sql
CREATE TABLE Drivers
(
    Driver_ID NUMBER(5) PRIMARY KEY,
    Certificate_ID VARCHAR2(6) REFERENCES Certificates(Certificate_ID),
    TDL_Number VARCHAR2(8) REFERENCES Train_Licence(TDL_Number),
    Start_Date DATE,
    Full_Name VARCHAR2(100) NOT NULL,
    DOB DATE,
    Address VARCHAR2(500),
    Phone_Number VARCHAR2(15),
    Email VARCHAR2(50)
)


CREATE TABLE Stations
(
    Station_Code VARCHAR2(3) PRIMARY KEY,
    Station_Name VARCHAR2(100) NOT NULL,
    Latitude NUMBER(11,9),
    Longitude NUMBER(11,9)
)
```



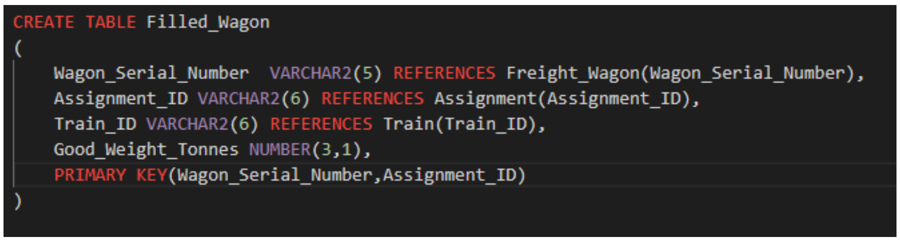
```
CREATE TABLE Stage
(
    Stage_ID NUMBER(10) PRIMARY KEY,
    Start_Station VARCHAR2(3) NOT NULL REFERENCES Stations(Station_Code),
    End_Station VARCHAR2(3) NOT NULL REFERENCES Stations(Station_Code),
    Distance NUMBER(5,2),
    CONSTRAINT Unique_Stage UNIQUE (Start_Station,End_Station)
)

CREATE TABLE Routes
(
    Route_ID NUMBER(10) PRIMARY KEY,
    Route_Name VARCHAR2(50)
)

CREATE TABLE Route_Builder
(
    Route_Builder_ID NUMBER(10),
    Stage_ID NUMBER(10) REFERENCES Stage(Stage_ID),
    Route_ID REFERENCES Routes(Route_ID),
    Route_Order NUMBER(2),
    PRIMARY KEY (Route_Builder_ID,Stage_ID)
)
```



```
CREATE TABLE Train
(
    Train_ID VARCHAR2(5) PRIMARY KEY,
    Loco_Serial_Number VARCHAR2(5) REFERENCES Locomotive(Serial_Number),
    Route_ID  NUMBER(10) REFERENCES Routes(Route_ID),
    Driver_ID NUMBER(5) REFERENCES Drivers(Driver_ID),
    Co_Driver_ID NUMBER(5) REFERENCES Drivers(Driver_ID),

    Departure_Time TIMESTAMP(0),
    Arrival_Time TIMESTAMP(0)
)

CREATE TABLE Company
(
    Company_ID NUMBER(5) PRIMARY KEY,
    Company_Name VARCHAR2(100) NOT NULL,
    Contact_Name VARCHAR2(100),
    Address VARCHAR2(3000),
    Phone_Number VARCHAR2(15),
    Email VARCHAR2(100)
)

CREATE TABLE Good_Type
(
    Good_ID NUMBER(10) PRIMARY KEY,
    Good_Name VARCHAR2(50),
    Good_Description VARCHAR2(3000),
    Unit_Weight_Grams NUMBER(5)
)
```



```
CREATE TABLE Collection_Delivery
(
    Collection_Delivery_ID NUMBER(10) PRIMARY KEY,
    Collection_station VARCHAR2(3) NOT NULL REFERENCES Stations(Station_Code),
    Delivery_Station VARCHAR2(3) NOT NULL REFERENCES Stations(Station_Code),
    CONSTRAINT Unique_CD UNIQUE(Collection_station,Delivery_Station)
)

CREATE TABLE Assignment
(
    Assignment_ID VARCHAR2(6) PRIMARY KEY,
    Company_ID NUMBER(5) REFERENCES Company(Company_ID),
    Collection_Delivery_ID NUMBER(10) REFERENCES Collection_Delivery(Collection_Delivery_ID)
)

CREATE TABLE Goods
(
    Assignment_ID VARCHAR2(6) REFERENCES Assignment(Assignment_ID),
    Good_ID NUMBER(10) REFERENCES Good_Type(Good_ID),
    Total_Weight_Tonnes NUMBER(5,1),
    PRIMARY KEY(Assignment_ID,Good_ID)
)
```

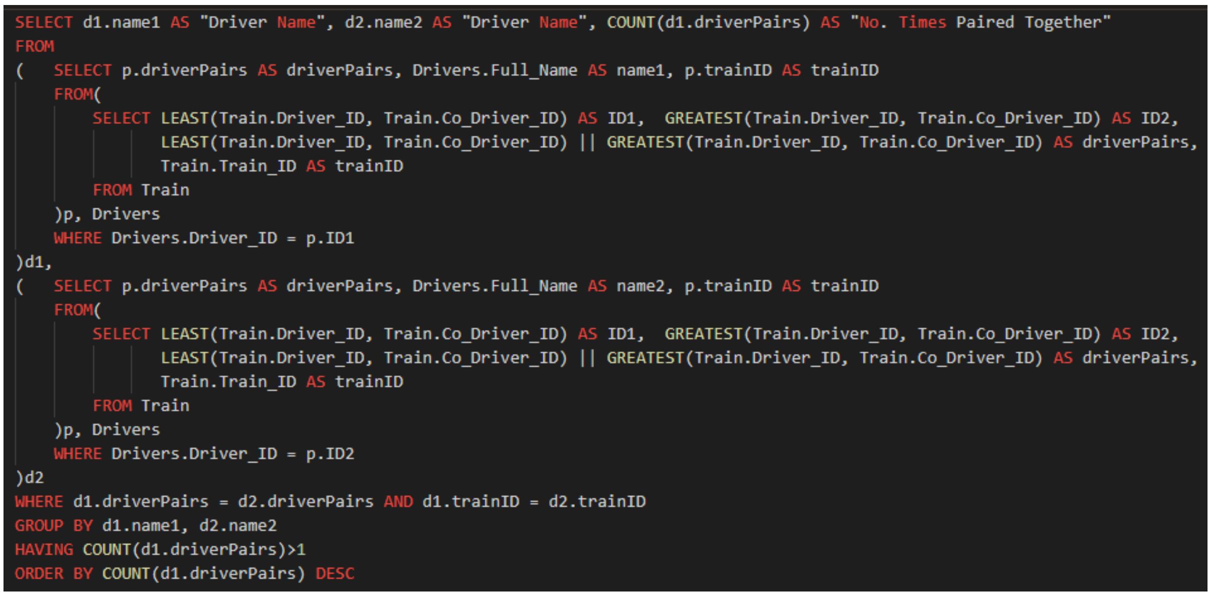


```
CREATE TABLE Filled_Wagon
(
    Wagon_Serial_Number  VARCHAR2(5) REFERENCES Freight_Wagon(Wagon_Serial_Number),
    Assignment_ID VARCHAR2(6) REFERENCES Assignment(Assignment_ID),
    Train_ID VARCHAR2(6) REFERENCES Train(Train_ID),
    Good_Weight_Tonnes NUMBER(3,1),
    PRIMARY KEY(Wagon_Serial_Number,Assignment_ID)
)
```

SQL Queries

Query 1

```
SELECT d1.name1 AS "Driver Name", d2.name2 AS "Driver Name", COUNT(d1.driverPairs) AS "No.
Times Paired Together"

FROM

(   SELECT p.driverPairs AS driverPairs, Drivers.Full_Name AS name1, p.trainID AS trainID

    FROM(

        SELECT LEAST(Train.Driver_ID, Train.Co_Driver_ID) AS ID1,
GREATEST(Train.Driver_ID, Train.Co_Driver_ID) AS ID2,

               LEAST(Train.Driver_ID, Train.Co_Driver_ID) || GREATEST(Train.Driver_ID,
Train.Co_Driver_ID) AS driverPairs,

               Train.Train_ID AS trainID

        FROM Train

    )p, Drivers

    WHERE Drivers.Driver_ID = p.ID1

)d1,

(   SELECT p.driverPairs AS driverPairs, Drivers.Full_Name AS name2, p.trainID AS trainID

    FROM(

        SELECT LEAST(Train.Driver_ID, Train.Co_Driver_ID) AS ID1,
GREATEST(Train.Driver_ID, Train.Co_Driver_ID) AS ID2,

               LEAST(Train.Driver_ID, Train.Co_Driver_ID) || GREATEST(Train.Driver_ID,
Train.Co_Driver_ID) AS driverPairs,

               Train.Train_ID AS trainID

        FROM Train

    )p, Drivers

    WHERE Drivers.Driver_ID = p.ID2

)d2

WHERE d1.driverPairs = d2.driverPairs AND d1.trainID = d2.trainID

GROUP BY d1.name1, d2.name2

HAVING COUNT(d1.driverPairs)>1

ORDER BY COUNT(d1.driverPairs) DESC
```



```
SELECT d1.name1 AS "Driver Name", d2.name2 AS "Driver Name", COUNT(d1.driverPairs) AS "No. Times Paired Together"
FROM
(   SELECT p.driverPairs AS driverPairs, Drivers.Full_Name AS name1, p.trainID AS trainID
    FROM(
        SELECT LEAST(Train.Driver_ID, Train.Co_Driver_ID) AS ID1,  GREATEST(Train.Driver_ID, Train.Co_Driver_ID) AS ID2,
               LEAST(Train.Driver_ID, Train.Co_Driver_ID) || GREATEST(Train.Driver_ID, Train.Co_Driver_ID) AS driverPairs,
               Train.Train_ID AS trainID
        FROM Train
    )p, Drivers
    WHERE Drivers.Driver_ID = p.ID1
)d1,
(   SELECT p.driverPairs AS driverPairs, Drivers.Full_Name AS name2, p.trainID AS trainID
    FROM(
        SELECT LEAST(Train.Driver_ID, Train.Co_Driver_ID) AS ID1,  GREATEST(Train.Driver_ID, Train.Co_Driver_ID) AS ID2,
               LEAST(Train.Driver_ID, Train.Co_Driver_ID) || GREATEST(Train.Driver_ID, Train.Co_Driver_ID) AS driverPairs,
               Train.Train_ID AS trainID
        FROM Train
    )p, Drivers
    WHERE Drivers.Driver_ID = p.ID2
)d2
WHERE d1.driverPairs = d2.driverPairs AND d1.trainID = d2.trainID
GROUP BY d1.name1, d2.name2
HAVING COUNT(d1.driverPairs)>1
ORDER BY COUNT(d1.driverPairs) DESC
```

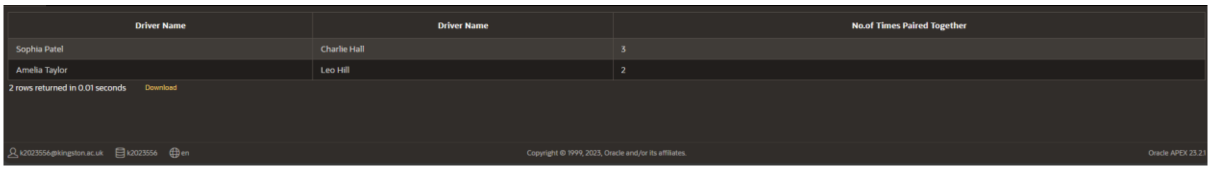
This shows the drivers who have driven together more than once. This SQL statement considers that the drivers could be either the main or the co-driver. It does this in the innermost select statements. Here, we look at the Driver and Co-driver columns and find which of the two IDs is smaller or greater. The smallest ID is then always concatenated with the larger ID to form a "driver pair". This "driverPair" will always be the same for the same two drivers regardless of who was co-driver or main.

As you can also see, the d1 and d2 statements are almost identical. I needed this to get the names of both drivers. So, if you look at d1, there is a SELECT statement which includes the field Drivers.Full_Name AS name1. This will be used later to get the name of the first driver. I set the WHERE clause to "Drivers.Driver_ID = p.ID1" to get this name. This is saying that the Driver_ID, for the name I need to get, is equal to the smaller two Diver_IDs in the pair. Thus, "Drivers.Full_Name AS name1" gets populated with the driver's full name and corresponding ID.

To get name2 or the name of the second driver of the pair, I duplicated d1 and set the WHERE clause as "Drivers.Driver_ID = p.ID2". This would set "Drivers.Full_Name AS name2" as the second driver of the pair as ID2 is equal to the larger ID number of the "driverPair".

Once I had this, the outer select statement was about getting both names and counting the driver pairs.

The result of the query is shown below:



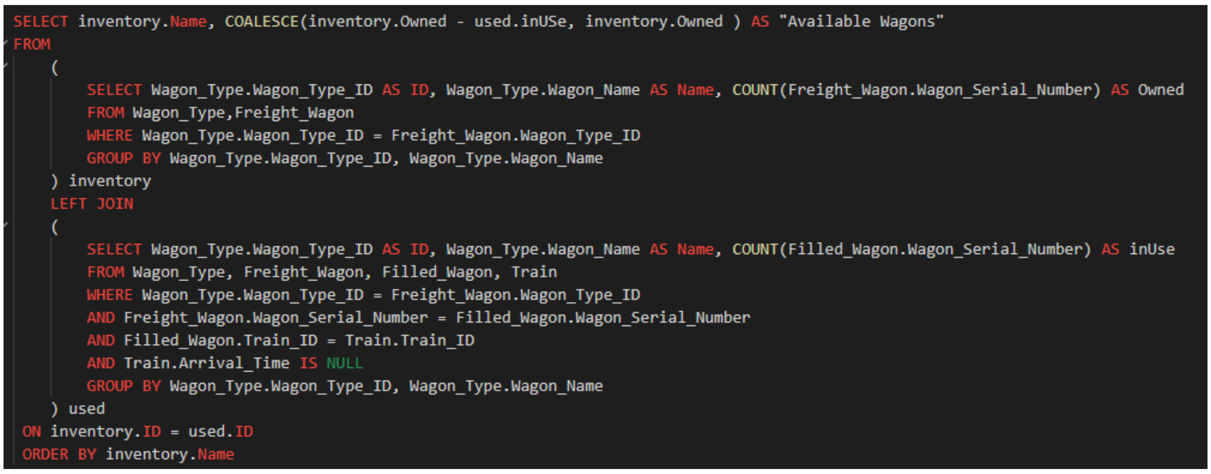
Driver Name	Driver Name	No.of Times Paired Together
Sophia Patel	Charlie Hall	3
Amelia Taylor	Leo Hill	2

2 rows returned in 0.01 seconds Download

k2023556@kingston.ac.uk k2023556 en Copyright © 1999, 2023, Oracle and/or its affiliates. Oracle APEX 23.2.1

Query 2

```
SELECT inventory.Name, COALESCE(inventory.Owned - used.inUSe, inventory.Owned ) AS
"Available Wagons"

FROM

    (

        SELECT Wagon_Type.Wagon_Type_ID AS ID, Wagon_Type.Wagon_Name AS Name,
COUNT(Freight_Wagon.Wagon_Serial_Number) AS Owned

        FROM Wagon_Type,Freight_Wagon

        WHERE Wagon_Type.Wagon_Type_ID = Freight_Wagon.Wagon_Type_ID

        GROUP BY Wagon_Type.Wagon_Type_ID, Wagon_Type.Wagon_Name

    ) inventory

    LEFT JOIN

    (

        SELECT Wagon_Type.Wagon_Type_ID AS ID, Wagon_Type.Wagon_Name AS Name,
COUNT(Filled_Wagon.Wagon_Serial_Number) AS inUse

        FROM Wagon_Type, Freight_Wagon, Filled_Wagon, Train

        WHERE Wagon_Type.Wagon_Type_ID = Freight_Wagon.Wagon_Type_ID

        AND Freight_Wagon.Wagon_Serial_Number = Filled_Wagon.Wagon_Serial_Number

        AND Filled_Wagon.Train_ID = Train.Train_ID

        AND Train.Arrival_Time IS NULL

        GROUP BY Wagon_Type.Wagon_Type_ID, Wagon_Type.Wagon_Name

    ) used

 ON inventory.ID = used.ID

 ORDER BY inventory.Name
```

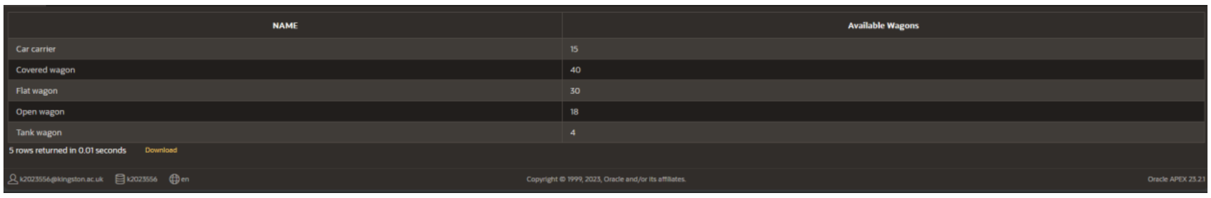


```
SELECT inventory.Name, COALESCE(inventory.Owned - used.inUSe, inventory.Owned ) AS "Available Wagons"
FROM
    (
        SELECT Wagon_Type.Wagon_Type_ID AS ID, Wagon_Type.Wagon_Name AS Name, COUNT(Freight_Wagon.Wagon_Serial_Number) AS Owned
        FROM Wagon_Type,Freight_Wagon
        WHERE Wagon_Type.Wagon_Type_ID = Freight_Wagon.Wagon_Type_ID
        GROUP BY Wagon_Type.Wagon_Type_ID, Wagon_Type.Wagon_Name
    ) inventory
    LEFT JOIN
    (
        SELECT Wagon_Type.Wagon_Type_ID AS ID, Wagon_Type.Wagon_Name AS Name, COUNT(Filled_Wagon.Wagon_Serial_Number) AS inUse
        FROM Wagon_Type, Freight_Wagon, Filled_Wagon, Train
        WHERE Wagon_Type.Wagon_Type_ID = Freight_Wagon.Wagon_Type_ID
        AND Freight_Wagon.Wagon_Serial_Number = Filled_Wagon.Wagon_Serial_Number
        AND Filled_Wagon.Train_ID = Train.Train_ID
        AND Train.Arrival_Time IS NULL
        GROUP BY Wagon_Type.Wagon_Type_ID, Wagon_Type.Wagon_Name
    ) used
  ON inventory.ID = used.ID
  ORDER BY inventory.Name
```

This SQL statement gives me the current stock level for all freight wagon types. The current time is defined by the query selecting the wagons used on trains that currently do not have an arrival time. The fact that they don't have an arrival time suggests that these train journeys are in progress. The table of currently used wagons is saved under the variable "used", while the total number of each freight wagon type is saved under the variable "inventory".

When running the query, the "used" table only had the inUse totals for two wagon types, meaning the other three wagon types did not show up in the table as none were in use. This posed a problem as the final table I want to produce should show all the freight wagon types and their respective stock levels. This was only possible because I used the COALESCE function. The COALESCE function acted like an IF statement where if there were an inUse value, this would be subtracted from the total number of the corresponding freight wagon type owned. At the same time, if the in-use value is null, then it is assumed that all the wagons of that type are available, so instead of subtracting, the query selects the total amount owned.

The result of the query is shown below:



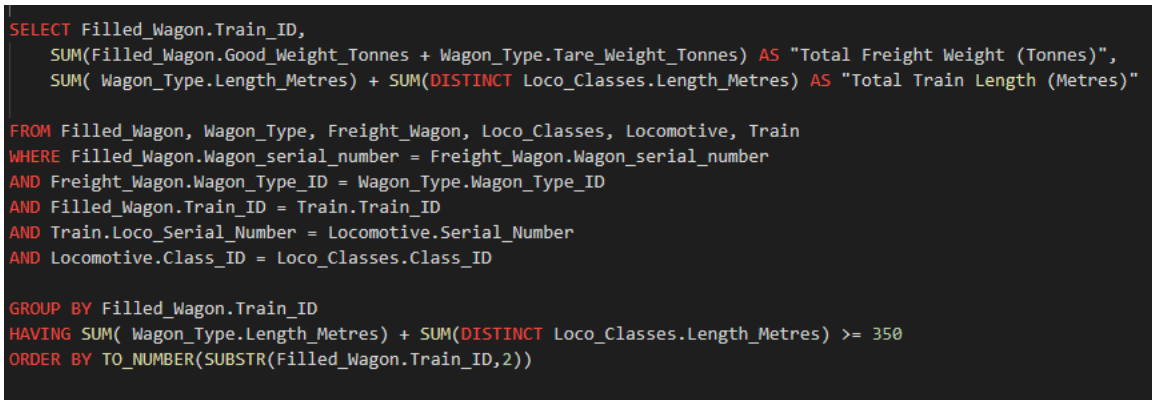
NAME	Available Wagons
Car carrier	15
Covered wagon	40
Flat wagon	30
Open wagon	18
Tank wagon	4

5 rows returned in 0.01 seconds Download

k2023556@kingston.ac.uk k2023556 en Copyright © 1999, 2023, Oracle and/or its affiliates. Oracle APEX 23.2.1

Query 3

```
SELECT Filled_Wagon.Train_ID,

    SUM(Filled_Wagon.Good_Weight_Tonnes + Wagon_Type.Tare_Weight_Tonnes) AS "Total Freight
Weight (Tonnes)",

    SUM( Wagon_Type.Length_Metres) + SUM(DISTINCT Loco_Classes.Length_Metres) AS "Total
Train Length (Metres)"



FROM Filled_Wagon, Wagon_Type, Freight_Wagon, Loco_Classes, Locomotive, Train

WHERE Filled_Wagon.Wagon_serial_number = Freight_Wagon.Wagon_serial_number

AND Freight_Wagon.Wagon_Type_ID = Wagon_Type.Wagon_Type_ID

AND Filled_Wagon.Train_ID = Train.Train_ID

AND Train.Loco_Serial_Number = Locomotive.Serial_Number

AND Locomotive.Class_ID = Loco_Classes.Class_ID



GROUP BY Filled_Wagon.Train_ID

HAVING SUM( Wagon_Type.Length_Metres) + SUM(DISTINCT Loco_Classes.Length_Metres) >= 350

ORDER BY TO_NUMBER(SUBSTR(Filled_Wagon.Train_ID,2))
```

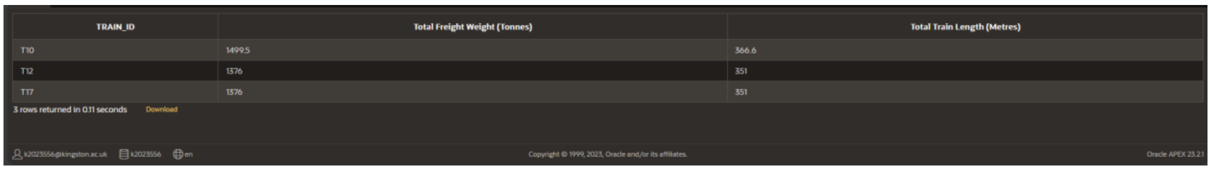


```
SELECT Filled_Wagon.Train_ID,
    SUM(Filled_Wagon.Good_Weight_Tonnes + Wagon_Type.Tare_Weight_Tonnes) AS "Total Freight Weight (Tonnes)",
    SUM( Wagon_Type.Length_Metres) + SUM(DISTINCT Loco_Classes.Length_Metres) AS "Total Train Length (Metres)"

FROM Filled_Wagon, Wagon_Type, Freight_Wagon, Loco_Classes, Locomotive, Train
WHERE Filled_Wagon.Wagon_serial_number = Freight_Wagon.Wagon_serial_number
AND Freight_Wagon.Wagon_Type_ID = Wagon_Type.Wagon_Type_ID
AND Filled_Wagon.Train_ID = Train.Train_ID
AND Train.Loco_Serial_Number = Locomotive.Serial_Number
AND Locomotive.Class_ID = Loco_Classes.Class_ID

GROUP BY Filled_Wagon.Train_ID
HAVING SUM( Wagon_Type.Length_Metres) + SUM(DISTINCT Loco_Classes.Length_Metres) >= 350
ORDER BY TO_NUMBER(SUBSTR(Filled_Wagon.Train_ID,2))
```

This SQL statement calculates the train lengths and weights. Then, filtering the results for the trains with a length exceeding 350

The Train_ID is a varchar with the format "T1". Because of this, the order is treated differently. Example: T1, T2, T10 would get ordered to T1, T10, T2. Instead of T1, T2, T10

Because of this ordering issue, I sliced off The T and cast the remaining digit as a number.



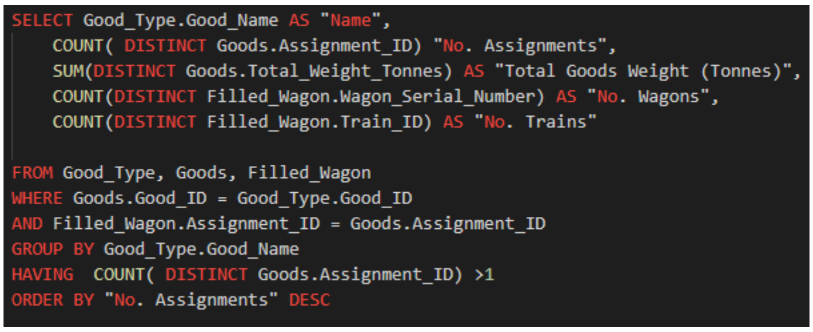
TRAIN_ID	Total Freight Weight (Tonnes)	Total Train Length (Metres)
T10	1499.5	366.6
T12	1376	351
T17	1376	351

3 rows returned in 0.11 seconds Download

k2023556@kingston.ac.uk k2023556 en Copyright © 1999, 2023, Oracle and/or its affiliates. Oracle APEX 23.2.1

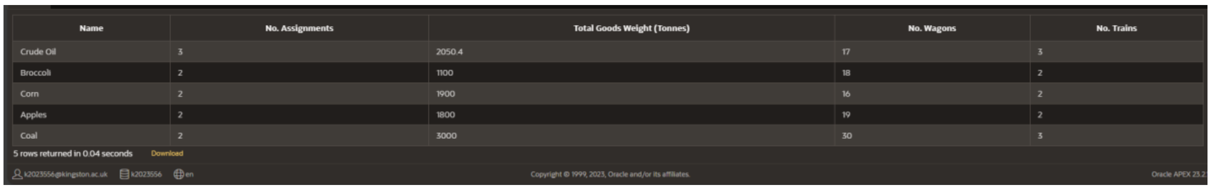
Query 4

```
SELECT Good_Type.Good_Name AS "Name",
    COUNT( DISTINCT Goods.Assignment_ID) "No. Assignments",
    SUM(DISTINCT Goods.Total_Weight_Tonnes) AS "Total Goods Weight (Tonnes)",
    COUNT(DISTINCT Filled_Wagon.Wagon_Serial_Number) AS "No. Wagons",
    COUNT(DISTINCT Filled_Wagon.Train_ID) AS "No. Trains"

FROM Good_Type, Goods, Filled_Wagon
WHERE Goods.Good_ID = Good_Type.Good_ID
AND Filled_Wagon.Assignment_ID = Goods.Assignment_ID
GROUP BY Good_Type.Good_Name
HAVING  COUNT( DISTINCT Goods.Assignment_ID) >1
ORDER BY "No. Assignments" DESC
```



```
SELECT Good_Type.Good_Name AS "Name",
    COUNT( DISTINCT Goods.Assignment_ID) "No. Assignments",
    SUM(DISTINCT Goods.Total_Weight_Tonnes) AS "Total Goods Weight (Tonnes)",
    COUNT(DISTINCT Filled_Wagon.Wagon_Serial_Number) AS "No. Wagons",
    COUNT(DISTINCT Filled_Wagon.Train_ID) AS "No. Trains"

FROM Good_Type, Goods, Filled_Wagon
WHERE Goods.Good_ID = Good_Type.Good_ID
AND Filled_Wagon.Assignment_ID = Goods.Assignment_ID
GROUP BY Good_Type.Good_Name
HAVING  COUNT( DISTINCT Goods.Assignment_ID) >1
ORDER BY "No. Assignments" DESC
```

This query shows which goods were ordered more than once and the resources/assets used to accomplish these orders. This could be useful if they wanted to compare how, over time, different types of goods required more or fewer assets and how much total weight could be moved with said assets.



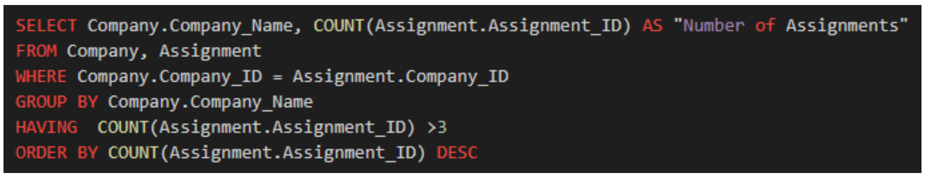
Name	No. Assignments	Total Goods Weight (Tonnes)	No. Wagons	No. Trains
Crude Oil	3	2050.4	17	3
Broccoli	2	1100	18	2
Corn	2	1900	16	2
Apples	2	1800	19	2
Coal	2	3000	30	3

5 rows returned in 0.04 seconds Download

k2023556@kingston.ac.uk k2023556 en Copyright © 1999, 2023, Oracle and/or its affiliates. Oracle APEX 23.2

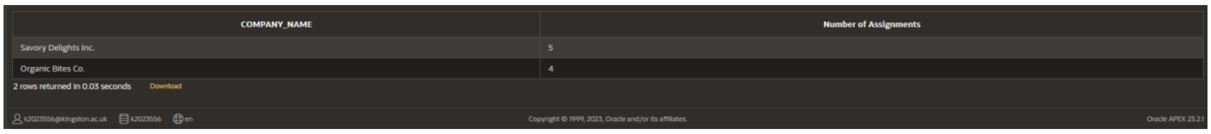
Query 5

```
SELECT Company.Company_Name, COUNT(Assignment.Assignment_ID) AS "Number of Assignments"

FROM Company, Assignment

WHERE Company.Company_ID = Assignment.Company_ID

GROUP BY Company.Company_Name

HAVING  COUNT(Assignment.Assignment_ID) >3

ORDER BY COUNT(Assignment.Assignment_ID) DESC
```



```
SELECT Company.Company_Name, COUNT(Assignment.Assignment_ID) AS "Number of Assignments"
FROM Company, Assignment
WHERE Company.Company_ID = Assignment.Company_ID
GROUP BY Company.Company_Name
HAVING  COUNT(Assignment.Assignment_ID) >3
ORDER BY COUNT(Assignment.Assignment_ID) DESC
```

Finding the companies that have more than three assignments



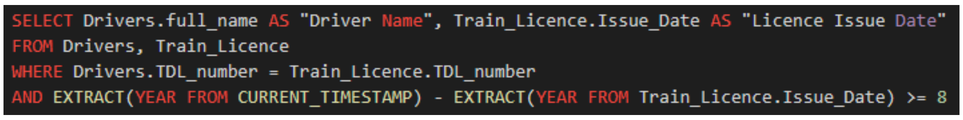
COMPANY_NAME	Number of Assignments
Savory Delights Inc.	5
Organic Bites Co.	4

2 rows returned in 0.03 seconds Download

k2023556@kingston.ac.uk k2023556 en Copyright © 1999, 2023, Oracle and/or its affiliates. Oracle APEX 23.2.1

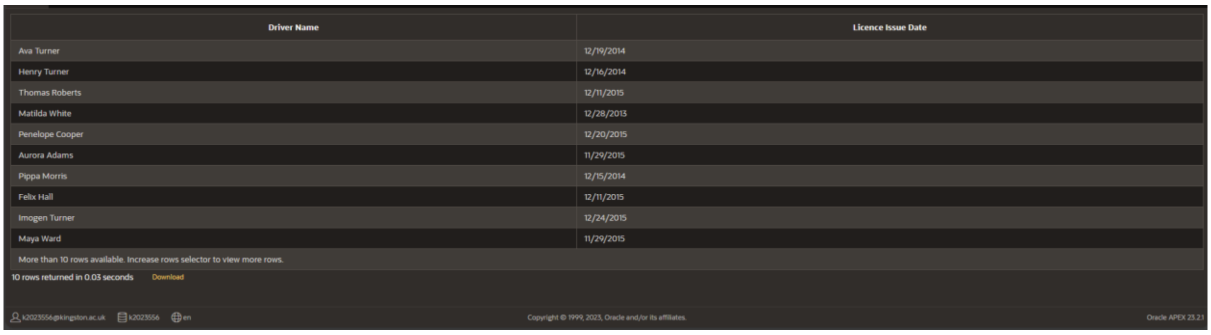
Query 6

```
SELECT Drivers.full_name AS "Driver Name", Train_Licence.Issue_Date AS "Licence Issue
Date"

FROM Drivers, Train_Licence

WHERE Drivers.TDL_number = Train_Licence.TDL_number

AND EXTRACT(YEAR FROM CURRENT_TIMESTAMP) - EXTRACT(YEAR FROM Train_Licence.Issue_Date) >=
8
```



```
SELECT Drivers.full_name AS "Driver Name", Train_Licence.Issue_Date AS "Licence Issue Date"
FROM Drivers, Train_Licence
WHERE Drivers.TDL_number = Train_Licence.TDL_number
AND EXTRACT(YEAR FROM CURRENT_TIMESTAMP) - EXTRACT(YEAR FROM Train_Licence.Issue_Date) >= 8
```

This query retrieves all the drivers with a train licence greater or equal to eight years old from the time the query is run.



Driver Name	Licence Issue Date
Ava Turner	12/19/2014
Henry Turner	12/16/2014
Thomas Roberts	12/11/2015
Matilda White	12/28/2013
Penelope Cooper	12/20/2015
Aurora Adams	11/29/2015
Pippa Morris	12/15/2014
Felix Hall	12/11/2015
Imogen Turner	12/24/2015
Maya Ward	11/29/2015

More than 10 rows available. Increase rows selector to view more rows.

10 rows returned in 0.03 seconds Download

k2023556@kingston.ac.uk k2023556 en Copyright © 1999, 2023, Oracle and/or its affiliates. Oracle APEX 23.2.1

Conclusion

Overall, the database which I have made is able to meet the needs of the SSR. For this conclusion I will summarize the steps I have taken to reach this point.

During the development of the system, the design stage was the first and most critical phase. I began by creating a class diagram for the entire system and periodically received feedback from my supervisor (teacher). This stage required me to evaluate the assignment brief and its requirements. From this class diagram, I created a relational diagram that helped me understand how each table would interact and relate to one another. I did this by explicitly stating the table an attribute references and its data type. While examining these relationships closely, I made minor adjustments to my class diagram, especially concerning attribute and table naming. Once I was satisfied with my relational diagram, I proceeded to create the tables in Oracle Apex.

I learned a valuable lesson when making the tables in Oracle Apex. Using Apex’s table creator is far more time-consuming than writing a table statement manually. The table creator doesn’t allow foreign keys and constraints to be added upon creation; they must be added after the table has been created. Furthermore, by writing the table statements, I can save them later in a separate document in case I need to re-make the database.

I had fun working on the project, but it did have a few frustrating moments. One of the challenges was generating the data required to test the implementation. While ChatGPT was helpful, it did face some difficulties when it had to provide data that required many constraints and references to other tables. The most time-consuming task was ensuring that the generated data was viable, which involved ensuring assignments were on the correct train and loaded onto the correct wagon without overloading it. Additionally, I had to ensure that the train did not exceed its maximum length and towing weight.

Throughout the process of importing data into the database, changes were made to the relational diagram, class diagram, and, subsequently, the table creation statements. These changes were necessary due to certain fields requiring either more space or a different data type. To avoid the problem of having to drop many tables just to change the structure or data type of a column to support the data that needs to be imported, I created and imported data into tables one at a time. This ensured a smooth and efficient data import process.

After setting up the tables and importing the data, I created six queries to demonstrate the database's functionality. These queries had to refer to multiple tables and be relevant to what the SSR may want to achieve with the system. Designing these queries required me to consult Apex’s documentation frequently, and at times I had to conduct my own research to better understand certain functions, such as extracting the year from a timestamp or converting a string to a number. This phase of the assignment was extremely satisfying because it was gratifying to see my database design being able to handle different complex SQL statements. It always made me proud of what I had accomplished.