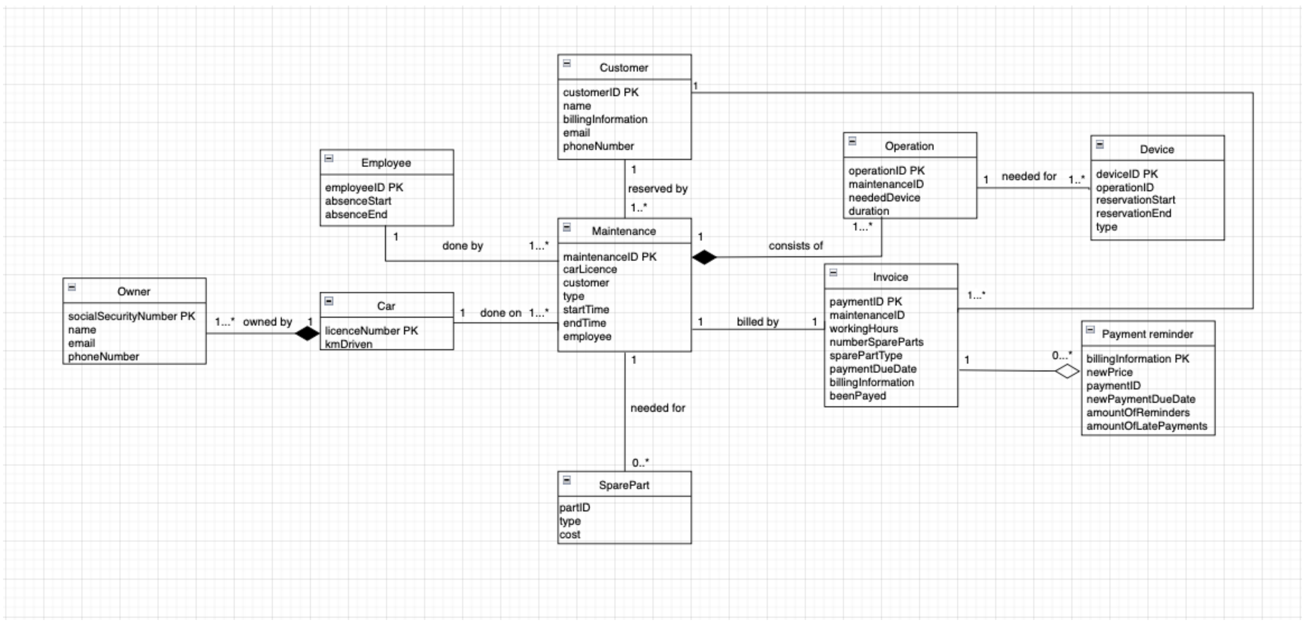
CS-A1150 Databaser

Group project: Garage

Date: 10.5.2022

Solution of the first part:



Relational data model:

Maintenances(maintenanceID, carLicence, customer, type, startTime, endTime, employee)

Customers(customerID, name, email, phoneNumber, billingInformation)

Employees(employeeID, absenceStart, absenceEnd)

Cars(licenceNumber, kmDriven)

Owners(socialSecurityNumber, carLicence, name, email, phoneNumber)

Operations(operationID, maintenanceID, neededDevice, duration)

Devices(deviceID, operationID, reservationStart, reservationEnd, type)

Invoices(paymentID, maintenanceID, workingHours, numberOfSpareParts, sparePartType, paymentDueDate, billingInformation, beenPayed)

PaymentRemainders(billingInformation, paymentID, newPrice, newPaymentDueDate, amountOfReminders, amountOfLatePayments)

SpareParts(partID, type, cost)

The functional dependencies:

maintenanceID → carLicence customer type startTime endTime employee

customerID → name email phoneNumber billingInformation employeeID → absenceStart absenceEnd

licenceNumber → kmDriven maintenanceID

socialSecurityNumber → carLicence name email phoneNumber

operationID → maintenanceID neededDevice duration

deviceID → operationID reservationStart reservationEnd type

invoiceID → maintenanceID workingHours numberOfSpareParts paymentDueDate billingInformation beenPayed

billingInformation → paymentID newPrice newPaymentDueDate amountOfReminders amountOfLatePayments

partID → type cost

Redundancies in our UML-diagram may be the contact information of the owner and customer because they may be the same person, but since it's possible that they are not we have two different relations for them.

Deletion anomalies exists in the Maintenance relation, where deleting a maintenance from the database deletes all other information that uses the maintenanceID as attribute also.

Checking all the closures we get from the functional dependencies for each relation.
maintenanceID: maintenanceID, carLicence, customer, type, startTime, endTime, employee
customerID: customerID, name, email, phoneNumber, billingInformation

employeeID: employeeID, absenceStart, absenceEnd

licenceNumber: licenceNumber, kmDriven, maintenanceID

socialSecurityNumber: socialSecurityNumber, carLicence, name, email, phoneNumber
operationID: operationID, maintenanceID, neededDevice, duration

deviceID: deviceID, operationID, reservationStart, reservationEnd, type

invoiceID: invoiceID, maintenanceID, workingHours, numberOfSpareParts, paymentDueDate, billingInformation, beenPayed

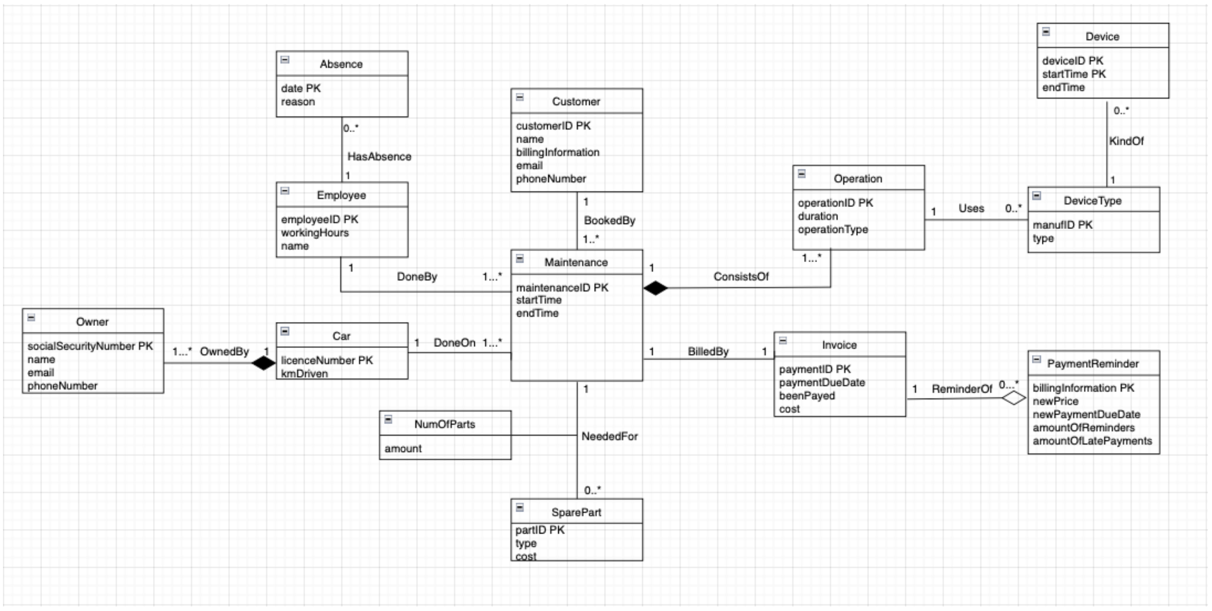
billingInformation: billingInformation, paymentID, newPrice, newPaymentDueDate, amountOfReminders, amountOfLatePayments

partID: partID, type, cost

Yes, it is in Boyce-Codd Normal Form because the closures contains all the attributes needed.

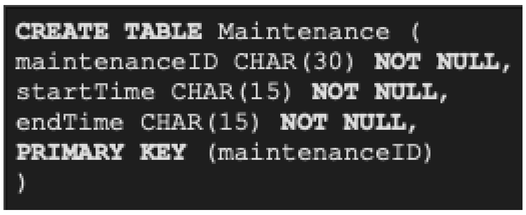
The modifications we have made are the following:

We added an attribute to know the operation type to the Operation class.
We added the class DeviceType to differentiate between the type and amount of devices.
We created an Abscess class, so that employees can have multiple abscess.
We added the class NumOfParts so that we will be able to count the amount of parts.

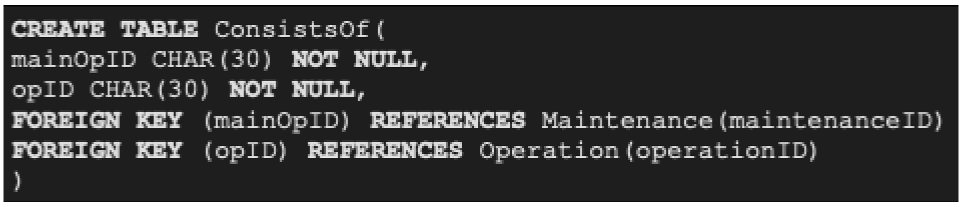


To make the database we created the following tables:

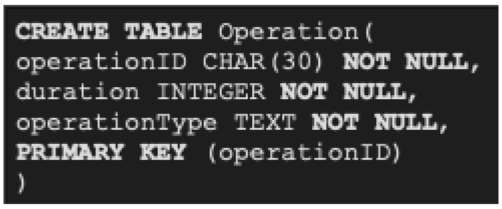
The primary keys of the classes consist usually of a single attribute. The foreign keys are a combination of the primary keys of the classes that are being connected. All the primary keys that work as an identification number have the value type CHAR. The other values are of type TEXT or INTEGER except for a single attribute that is of type BOOLEAN. Most of the values should not be of type NULL, except for some attributes that do not necessarily need a value, i.e reason in class Absence and kmDriven in class Car.



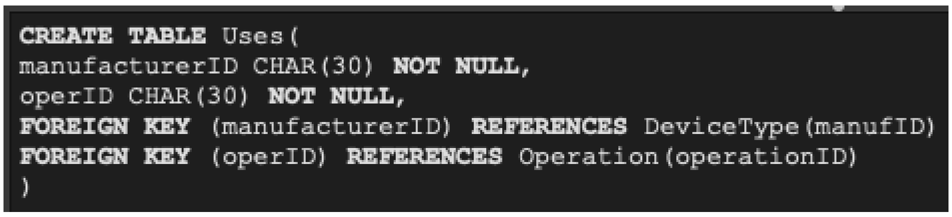
```
CREATE TABLE Maintenance (
maintenanceID CHAR(30) NOT NULL,
startTime CHAR(15) NOT NULL,
endTime CHAR(15) NOT NULL,
PRIMARY KEY (maintenanceID)
)
```



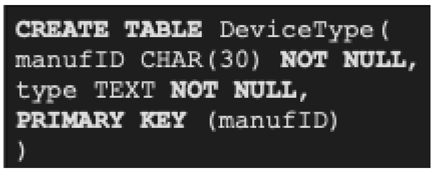
```
CREATE TABLE ConsistsOf(
mainOpID CHAR(30) NOT NULL,
opID CHAR(30) NOT NULL,
FOREIGN KEY (mainOpID) REFERENCES Maintenance(maintenanceID)
FOREIGN KEY (opID) REFERENCES Operation(operationID)
)
```



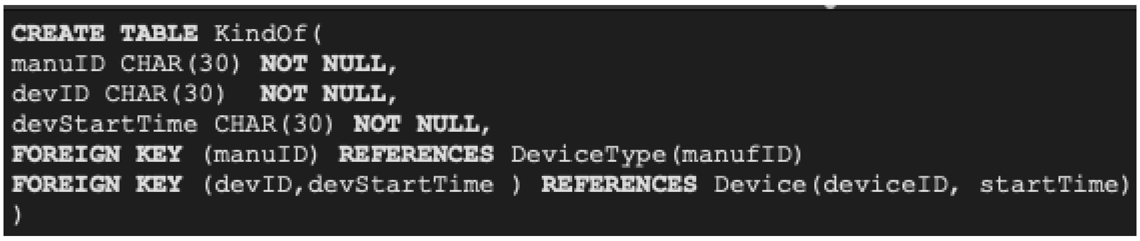
```
CREATE TABLE Operation(
operationID CHAR(30) NOT NULL,
duration INTEGER NOT NULL,
operationType TEXT NOT NULL,
PRIMARY KEY (operationID)
)
```



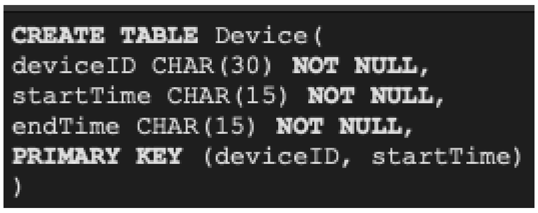
```
CREATE TABLE Uses(
manufacturerID CHAR(30) NOT NULL,
operID CHAR(30) NOT NULL,
FOREIGN KEY (manufacturerID) REFERENCES DeviceType(manufID)
FOREIGN KEY (operID) REFERENCES Operation(operationID)
)
```



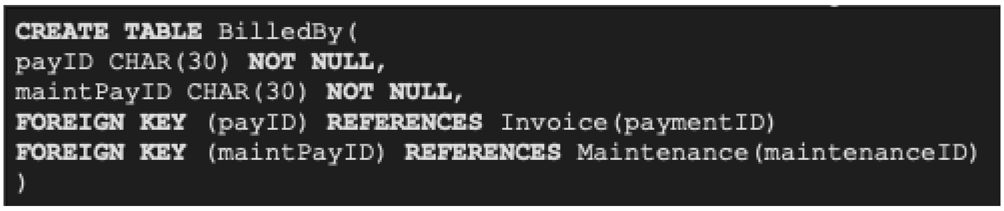
```
CREATE TABLE DeviceType(
manufID CHAR(30) NOT NULL,
type TEXT NOT NULL,
PRIMARY KEY (manufID)
)
```



```
CREATE TABLE KindOf(
manuID CHAR(30) NOT NULL,
devID CHAR(30)  NOT NULL,
devStartTime CHAR(30) NOT NULL,
FOREIGN KEY (manuID) REFERENCES DeviceType(manufID)
FOREIGN KEY (devID,devStartTime ) REFERENCES Device(deviceID, startTime)
)
```



```
CREATE TABLE Device(
deviceID CHAR(30) NOT NULL,
startTime CHAR(15) NOT NULL,
endTime CHAR(15) NOT NULL,
PRIMARY KEY (deviceID, startTime)
)
```



```
CREATE TABLE BilledBy(
payID CHAR(30) NOT NULL,
maintPayID CHAR(30) NOT NULL,
FOREIGN KEY (payID) REFERENCES Invoice(paymentID)
FOREIGN KEY (maintPayID) REFERENCES Maintenance(maintenanceID)
)
```

```sql
CREATE TABLE Invoice(
paymentID CHAR(30) NOT NULL,
paymentDueDate CHAR(10) NOT NULL,
beenPayed BOOLEAN NOT NULL,
costOfWork INTEGER NOT NULL,
PRIMARY KEY (paymentID)
)
```

```sql
CREATE TABLE RemainderOf(
remPayID CHAR(30) NOT NULL,
billInfo CHAR(30) NOT NULL,
FOREIGN KEY (remPayID) REFERENCES Invoice(paymentID),
FOREIGN KEY (billInfo) REFERENCES PaymentRemainder(billingInformation)
)
```

```sql
CREATE TABLE PaymentRemainder(
billingInformation CHAR(30) NOT NULL,
newPrice INTEGER NOT NULL,
newPaymentDueDate CHAR(10) NOT NULL,
amountOfRemainders INTEGER NOT NULL,
amountOfLatePayments INTEGER NOT NULL,
PRIMARY KEY (billingInformation)
)
```

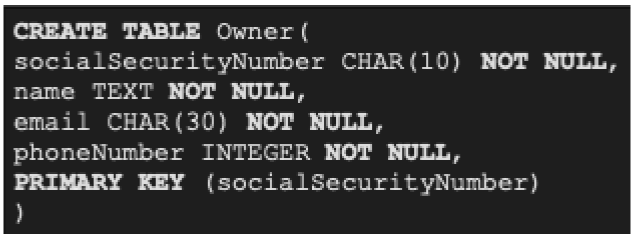
```sql
CREATE TABLE NeededFor(
spareID CHAR(30) NOT NULL,
maintSpareID CHAR(30) NOT NULL,
amount INTEGER,
FOREIGN KEY (spareID) REFERENCES SparePart(partID)
FOREIGN KEY (maintSpareID) REFERENCES Maintenance(maintenanceID)
)
```

```sql
CREATE TABLE SparePart(
partID CHAR(30) NOT NULL,
type TEXT NOT NULL,
cost INTEGER NOT NULL,
PRIMARY KEY (partID)
)
```

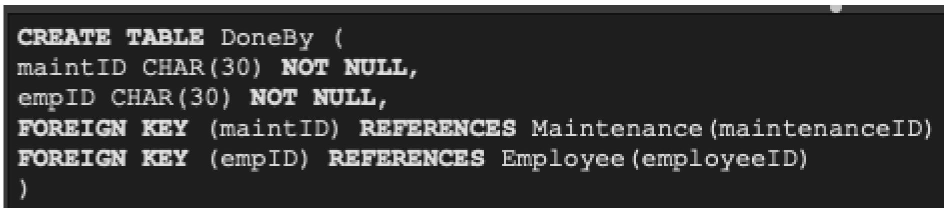
```sql
CREATE TABLE DoneOn(
licenceNr CHAR(30) NOT NULL,
mID CHAR(30) NOT NULL,
FOREIGN KEY (licenceNr) REFERENCES Car(licenceNumber)
FOREIGN KEY (mID) REFERENCES Maintenance(maintenanceID)
)
```

```sql
CREATE TABLE Car(
licenceNumber CHAR(30) NOT NULL,
kmDriven INTEGER,
PRIMARY KEY (licenceNumber)
)
```

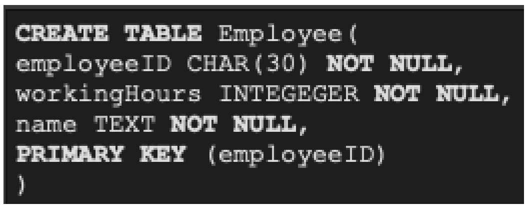
```sql
CREATE TABLE OwnedBy(
ssnr CHAR(10) NOT NULL,
licNr CHAR(10) NOT NULL,
FOREIGN KEY (ssnr) REFERENCES Owner(socialSecurityNumber)
FOREIGN KEY (licNr) REFERENCES Car(licenceNumber)
)
```



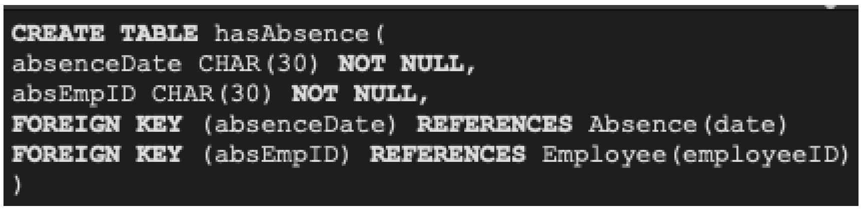
```sql
CREATE TABLE Owner(
socialSecurityNumber CHAR(10) NOT NULL,
name TEXT NOT NULL,
email CHAR(30) NOT NULL,
phoneNumber INTEGER NOT NULL,
PRIMARY KEY (socialSecurityNumber)
)
```



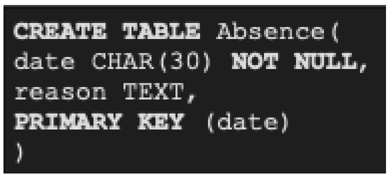
```sql
CREATE TABLE DoneBy (
maintID CHAR(30) NOT NULL,
empID CHAR(30) NOT NULL,
FOREIGN KEY (maintID) REFERENCES Maintenance(maintenanceID)
FOREIGN KEY (empID) REFERENCES Employee(employeeID)
)
```



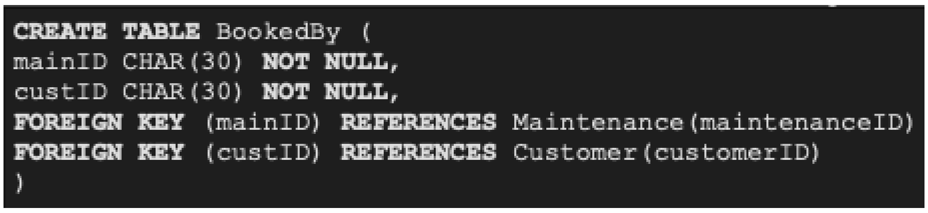
```sql
CREATE TABLE Employee(
employeeID CHAR(30) NOT NULL,
workingHours INTEGEGER NOT NULL,
name TEXT NOT NULL,
PRIMARY KEY (employeeID)
)
```



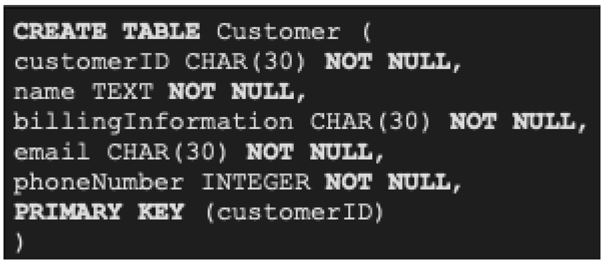
```sql
CREATE TABLE hasAbsence(
absenceDate CHAR(30) NOT NULL,
absEmpID CHAR(30) NOT NULL,
FOREIGN KEY (absenceDate) REFERENCES Absence(date)
FOREIGN KEY (absEmpID) REFERENCES Employee(employeeID)
)
```



```sql
CREATE TABLE Absence(
date CHAR(30) NOT NULL,
reason TEXT,
PRIMARY KEY (date)
)
```



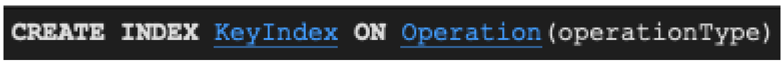
```sql
CREATE TABLE BookedBy (
mainID CHAR(30) NOT NULL,
custID CHAR(30) NOT NULL,
FOREIGN KEY (mainID) REFERENCES Maintenance(maintenanceID)
FOREIGN KEY (custID) REFERENCES Customer(customerID)
)
```



```sql
CREATE TABLE Customer (
customerID CHAR(30) NOT NULL,
name TEXT NOT NULL,
billingInformation CHAR(30) NOT NULL,
email CHAR(30) NOT NULL,
phoneNumber INTEGER NOT NULL,
PRIMARY KEY (customerID)
)
```

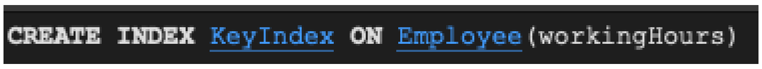
Queries that are typical for our database will most likely have something to do with customer information, employee or device availability or checking history of the garage. The database will be able to be used to find information about the customers, e.g who the owner(s) of the car is/are and if they are the same person who brought the car into the garage. The database will also help with the planning of which employee will do the reparation and with which device, by searching in the database who is available and when the devices are free. The database will also collect data on the cars and customers who have been at the garage.

Indexing is created automatically in SQLite for primary keys when a table is created and since our database consists of many small tables with a single primary key in each of them, we didn't see a reason to create too many other indices. Firstly we made an index for operationType, because the same kind of operation is often done during a maintenance session.



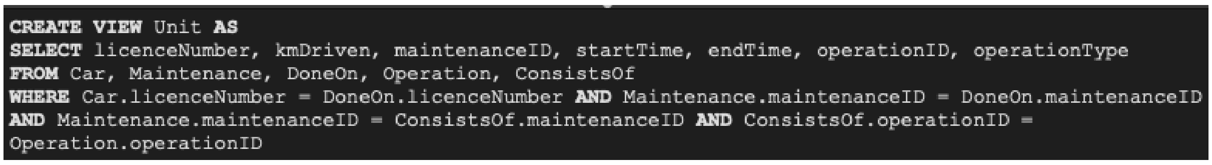
```
CREATE INDEX KeyIndex ON Operation(operationType)
```

and secondly we have workingHours, this might be useful so we can easily see how many hours each employee has worked during the week, which might be helpful when later on counting how much the workers get paid. The workingHours index can also be helpful to use if you want to count how many hours all the employees spent collectively working in the garage.

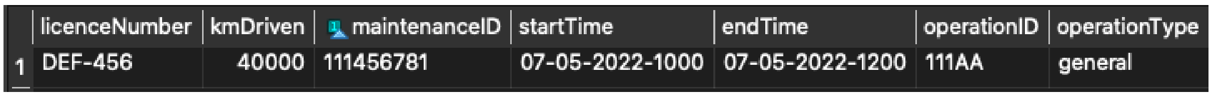


```
CREATE INDEX KeyIndex ON Employee(workingHours)
```

A view we saw useful to make was between the Car, Operation and Maintenance classes. The reason this view is useful to us is because we can now easily tell which car has had which operation done and when the car was in the garage.

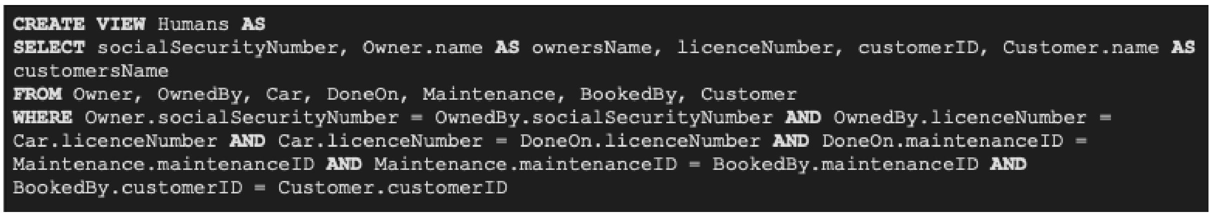


```
CREATE VIEW Unit AS
SELECT licenceNumber, kmDriven, maintenanceID, startTime, endTime, operationID, operationType
FROM Car, Maintenance, DoneOn, Operation, ConsistsOf
WHERE Car.licenceNumber = DoneOn.licenceNumber AND Maintenance.maintenanceID = DoneOn.maintenanceID
AND Maintenance.maintenanceID = ConsistsOf.maintenanceID AND ConsistsOf.operationID =
Operation.operationID
```

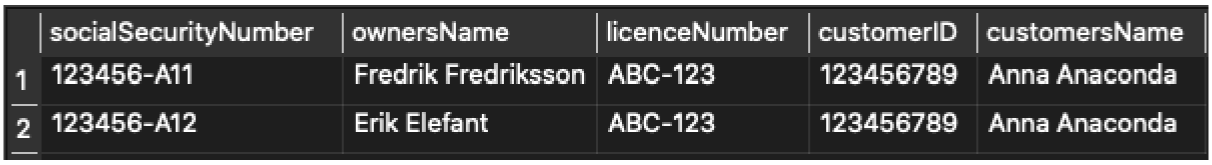


	licenceNumber	kmDriven	maintenanceID	startTime	endTime	operationID	operationType
1	DEF-456	40000	111456781	07-05-2022-1000	07-05-2022-1200	111AA	general

Another useful view might be one where you can see the car, their owners and who brought them in.

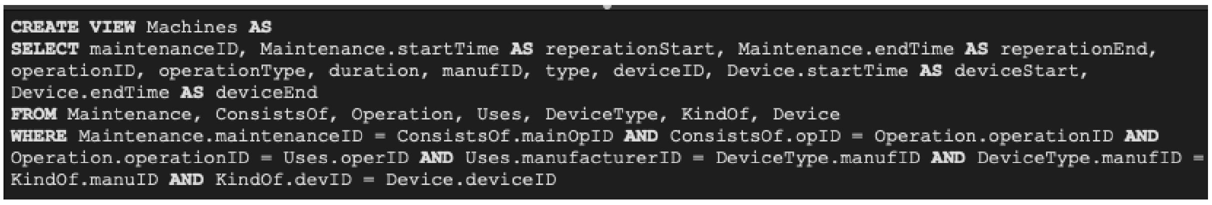


```
CREATE VIEW Humans AS
SELECT socialSecurityNumber, Owner.name AS ownersName, licenceNumber, customerID, Customer.name AS
customersName
FROM Owner, OwnedBy, Car, DoneOn, Maintenance, BookedBy, Customer
WHERE Owner.socialSecurityNumber = OwnedBy.socialSecurityNumber AND OwnedBy.licenceNumber =
Car.licenceNumber AND Car.licenceNumber = DoneOn.licenceNumber AND DoneOn.maintenanceID =
Maintenance.maintenanceID AND Maintenance.maintenanceID = BookedBy.maintenanceID AND
BookedBy.customerID = Customer.customerID
```

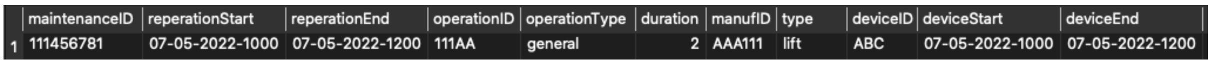


	socialSecurityNumber	ownersName	licenceNumber	customerID	customersName
1	123456-A11	Fredrik Fredriksson	ABC-123	123456789	Anna Anaconda
2	123456-A12	Erik Elefant	ABC-123	123456789	Anna Anaconda

The last view we decided to implement is one that connects the classes Maintenance, Operation, DeviceType and Device. This will make it more clear to see which device is used in what operation and how long it will be used for.



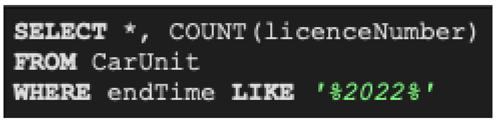
```
CREATE VIEW Machines AS
SELECT maintenanceID, Maintenance.startTime AS reperationStart, Maintenance.endTime AS reperationEnd,
operationID, operationType, duration, manufID, type, deviceID, Device.startTime AS deviceStart,
Device.endTime AS deviceEnd
FROM Maintenance, ConsistsOf, Operation, Uses, DeviceType, KindOf, Device
WHERE Maintenance.maintenanceID = ConsistsOf.mainOpID AND ConsistsOf.opID = Operation.operationID AND
Operation.operationID = Uses.operID AND Uses.manufacturerID = DeviceType.manufID AND DeviceType.manufID =
KindOf.manuID AND KindOf.devID = Device.deviceID
```



	maintenanceID	reperationStart	reperationEnd	operationID	operationType	duration	manufID	type	deviceID	deviceStart	deviceEnd
1	111456781	07-05-2022-1000	07-05-2022-1200	111AA	general	2	AAA111	lift	ABC	07-05-2022-1000	07-05-2022-1200

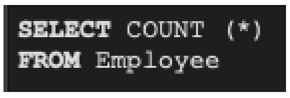
The queries we have implemented are the following:

1. Shows how many cars have been in the garage this year. This query counts how many cars have been in the garage during this year.



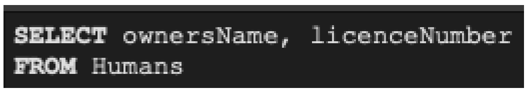
```
SELECT *, COUNT(licenceNumber)
FROM CarUnit
WHERE endTime LIKE '%2022%'
```

2. Search for the amount of employees who work at the garage. Counts how many employees have worked in the garage.



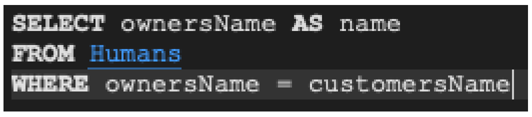
```
SELECT COUNT (*)
FROM Employee
```

3. Who owns the car. Selects the owner(s) of the car



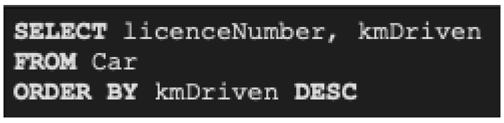
```
SELECT ownersName, licenceNumber
FROM Humans
```

4. Same person brought in the car and owns the car (doesn't return any rows if none of the customers that brought the car is the car owner). Check if the person who brought in the car is the same person as the one who owns it.



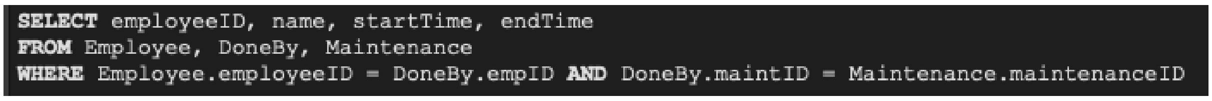
```
SELECT ownersName AS name
FROM Humans
WHERE ownersName = customersName
```

5. Cars ranked by driven km. Shows you how many km every car has driven and ranks them in descending order.

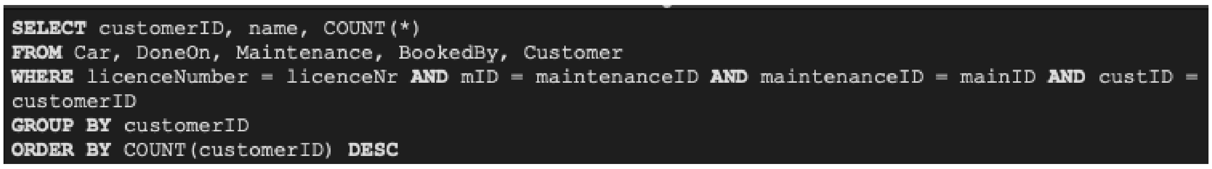


```
SELECT licenceNumber, kmDriven
FROM Car
ORDER BY kmDriven DESC
```

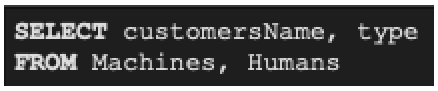
6. When a certain employee is free. Returns a list that says when an employee is working so that you can based on that see when the employee is free and book them.


```sql
SELECT employeeID, name, startTime, endTime
FROM Employee, DoneBy, Maintenance
WHERE Employee.employeeID = DoneBy.empID AND DoneBy.maintID = Maintenance.maintenanceID
```

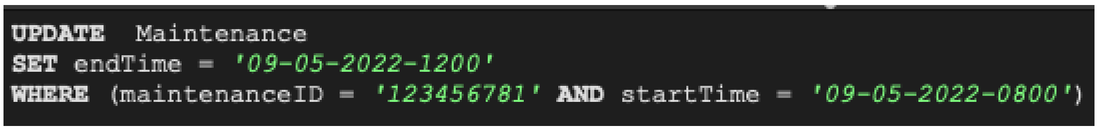
7. Rank customers by who has been there the most often. Ranks the customers in descending order by who has been there the most often. We group the tables by customerID so that we can count how many times one customer has visited the garage.


```sql
SELECT customerID, name, COUNT(*)
FROM Car, DoneOn, Maintenance, BookedBy, Customer
WHERE licenceNumber = licenceNr AND mID = maintenanceID AND maintenanceID = mainID AND custID =
customerID
GROUP BY customerID
ORDER BY COUNT(customerID) DESC
```

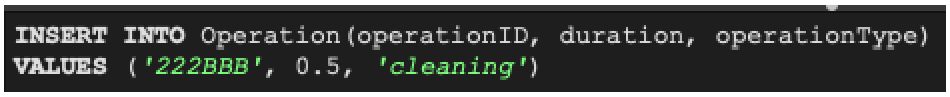
8. List which device was used on which customer's car. Uses two views to list the devices that have been used on a certain customer's cars.

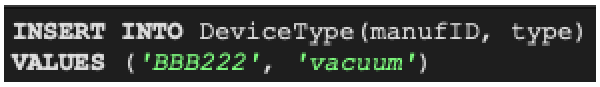

```sql
SELECT customersName, type
FROM Machines, Humans
```

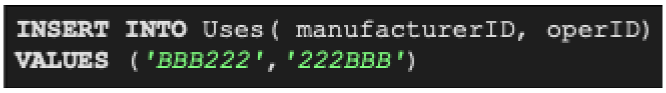
9. Update the end time for a reparation. Updates the end of a reparation time, due to delays.


```sql
UPDATE  Maintenance
SET endTime = '09-05-2022-1200'
WHERE (maintenanceID = '123456781' AND startTime = '09-05-2022-0800')
```

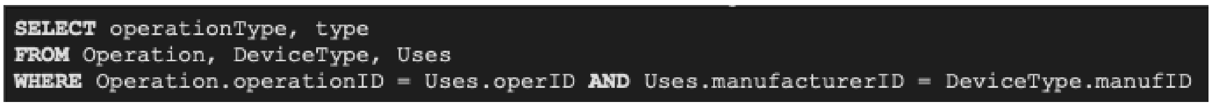
10. Insert new operation cleaning and needed device vacuum. The new operation is inserted and the new device is also inserted. We also have to insert it into the junction table to get it to work properly.

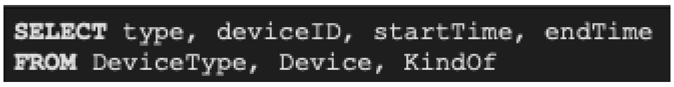

```sql
INSERT INTO Operation(operationID, duration, operationType)
VALUES ('222BBB', 0.5, 'cleaning')
```

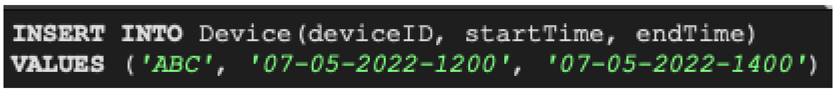

```sql
INSERT INTO DeviceType(manufID, type)
VALUES ('BBB222', 'vacuum')
```


```sql
INSERT INTO Uses( manufacturerID, operID)
VALUES ('BBB222','222BBB')
```

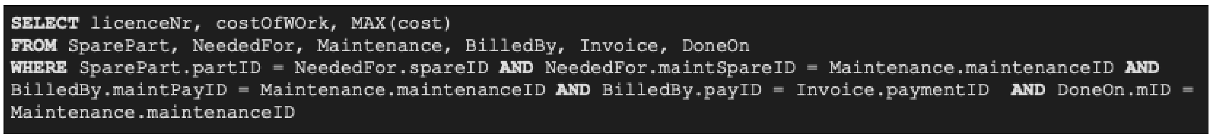
11. Match operation to a device when the device is free. Check what devices are needed for the reparation. Checks when one of the needed devices is available and books it for the time needed.


```sql
SELECT operationType, type
FROM Operation, DeviceType, Uses
WHERE Operation.operationID = Uses.operID AND Uses.manufacturerID = DeviceType.manufID
```


```sql
SELECT type, deviceID, startTime, endTime
FROM DeviceType, Device, KindOf
```

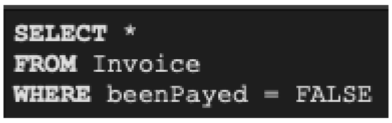

```sql
INSERT INTO Device(deviceID, startTime, endTime)
VALUES ('ABC', '07-05-2022-1200', '07-05-2022-1400')
```

12. Search for the car that has had the most expensive maintenance during the year. Returns the car that has had the most expensive maintenance done during the year.



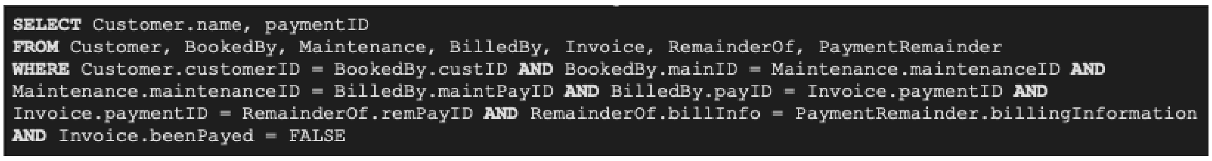
```
SELECT licenceNr, costOfWOrk, MAX(cost)
FROM SparePart, NeededFor, Maintenance, BilledBy, Invoice, DoneOn
WHERE SparePart.partID = NeededFor.spareID AND NeededFor.maintSpareID = Maintenance.maintenanceID AND
BilledBy.maintPayID = Maintenance.maintenanceID AND BilledBy.payID = Invoice.paymentID  AND DoneOn.mID =
Maintenance.maintenanceID
```

13. Search the unpaid invoices. Returns all the Invoices that are unpaid.



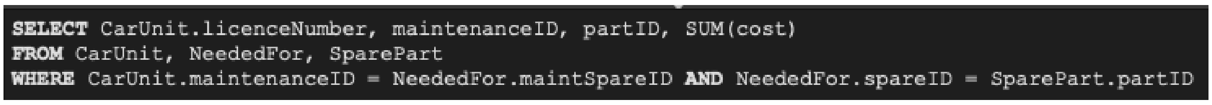
```
SELECT *
FROM Invoice
WHERE beenPayed = FALSE
```

14. Search for customers that have been sent invoices and haven’t paid the invoice. Returns the customers that haven’t paid their invoices.



```
SELECT Customer.name, paymentID
FROM Customer, BookedBy, Maintenance, BilledBy, Invoice, RemainderOf, PaymentRemainder
WHERE Customer.customerID = BookedBy.custID AND BookedBy.mainID = Maintenance.maintenanceID AND
Maintenance.maintenanceID = BilledBy.maintPayID AND BilledBy.payID = Invoice.paymentID AND
Invoice.paymentID = RemainderOf.remPayID AND RemainderOf.billInfo = PaymentRemainder.billingInformation
AND Invoice.beenPayed = FALSE
```

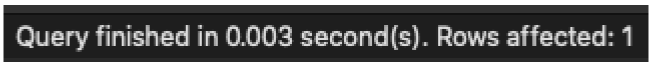
15. Sums up the cost of the different spare parts for the repair of a certain car. Returns the cost for the car's spare parts.



```
SELECT CarUnit.licenceNumber, maintenanceID, partID, SUM(cost)
FROM CarUnit, NeededFor, SparePart
WHERE CarUnit.maintenanceID = NeededFor.maintSpareID AND NeededFor.spareID = SparePart.partID
```

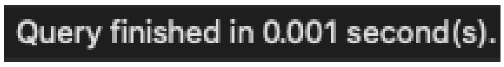
A lot of the queries became a lot easier to do, due to the views we implemented and therefore might not seem so difficult to make.

All of our queries have one of the two outputs. Both tells us that the query was finished and works how it should and the other one (where a tuple was inserted) tells how many rows were affected:



```
Query finished in 0.003 second(s). Rows affected: 1
```

,



```
Query finished in 0.001 second(s).
```

.

We didn’t get the Garage.sql file to open in SQLite (but it worked in VS Code) when we tried to test if the file opens after being exported, so we included the whole db.-file as well in case there would be problems.