02170 Mandatory Group Project

Albert Frisch Møller$^{[s214610]}$, Gustav Grønvold$^{[s184205]}$, Mark Andrawes$^{[s214654]}$, Tobias Holtoft Møller$^{[s184217]}$, and Ying Qi, Tiffanie Leong$^{[236763]}$

Technical University of Denmark
07/04/2024

Table of Contents

1 Statement of Requirements

The chosen database management system models a car reparations company. The company has various customers, and each customer owns a car with a specific set of characteristics, like brand and color. These cars are the subjects of repair job requests, which contain a brief description about what needs to be repaired, as well as a time slot (start time and end time) for the repair job. Additionally, each request contains a list of the tools and materials required for the repair job. Each repair job is assigned to a mechanic, where each mechanic has an assigned salary and works in a particular workshop. Each workshop has a maximum capacity of cars that can be handled at once, as well as a current capacity. Furthermore, each workshop has an inventory of tools and materials which are used to repair the cars.

The purpose of this database is to be able to effectively manage the repair job requests, and ensuring that the requests are allocated to workshops that contain enough space and the necessary inventory to complete the job.

2 Conceptual Design

Our model contains 6 entities which were briefly described in the statement of requirements. These entities are *Car*, *Customer*, *RepairJob*, *Mechanic*, *Workshops*, and *Inventory*. Between these entities lies many relations, both the entities and the relations will be described further in this section.

2.1 Entity descriptions

A repeating feature for all the entities is that they all have a unique identifier, which is used as foreign keys between most of our relations.

The RepairJob Entity is meant to model the repair job, and therefore ties all the other entities together. The job needs a *car* to work on. There should be a *customer* who places the order and pays for it. The Job is then assigned to a *mechanic* based on which *workshop* the job was created at. All these relations are modeled Fig. 1. Apart from these relations, the *RepairJob* contains the attributes *JobID*, *RepairDescription*, *StartTime*, and*EndTime*. of these attributes, only the JobID is important for the rest of the model, the tree other attributes only exists to satisfy the need of the real world example we are modeling.

The Car Entity is modeling the car that is being repaired, and as such it has two relations, one with the *RepairJob* and one with the *Customer*. Just like for the *RepairJob* entity, this entity contains multiple attributes that serve no other purpose than to satisfy the real-world example we are modeling. These being *color*, *LastServiced*, *Price*, and *Mileage*. The two last attributes *CarID* and *Brand* is used other places in the model, the *CarID* as a foreign key, and the brand is used for making sure that the mechanic working on the repair job knows how to repair cars of the given brand.

The Customer Entity is modeling the customer that owns the *Car* and that is requesting a *RepairJob*. The only attribute being used in the rest of the model is *CustomerID* as this is used as a foreign key. The other value *ContactInfo* is just to represent a real database where customers can be contacted when the repair job is finished.

The Mechanic Entity is the mechanic working at a *Workshop* and that is working on different *RepairJob*s. As described in the *Car* entity, the attribute *Brand* is used to select which jobs the mechanic is allowed to work on, apart from this, the only attribute being used by the rest of the model is *MechanicID* as this is used as a foreign key.

The Workshops Entity is the place where the *Car* is delivered to be repaired, and the workplace of the *Mechanic*. Our model also contains an Inventory that contains information about which items are in the different workshops. Apart from the attribute WorkshopID which is the primary key and is used for foreign keys, we also have two attributes *CurrCapacity* and *MaxCapacity*, which is modeling how many car repairs there is space for in each workshop, and how many cars are currently being repaired in each given workshop. If *CurrCapacity* is equal to *MaxCapacity* it means that the given workshop is full, and it shouldn't be able to take on any more jobs.

The Inventory Entity is modeling a collected inventory for all the *Workshops*, so that you can see all the items that exist, and which *Workshops* they are at. The *Inventory* entity contains an *ItemID*, which is the primary key together with *WorkshopID* which is a foreign key from the textitWorkshops entity. This will ensure that it's possible to have the same items stored in different *Workshops*. The attributes *Quantity* and *ItemName* doesn't have any relations with the rest of the model, except that as items are used for *RepairJob*s the quantity in the inventory is meant to be reduced.

2.2 Relation descriptions

Throughout our model, we have several relations between the different entities, these being *Owns*, *Subject of*, *Requests*, *Works on*, *Works at*, *Happens in*, and *Contains*. All the relations will be described further in the following section.
Owns is the relation between *Car* and *Customer*. The relation is a Many-to-one relation as a customer can own many cars, furthermore both entities have total participation as it wouldn't make sense if a customer didn't have a car, nor if a car didn't have a owner requesting the RepairJob.

Subject of is the relation between *Car* and *RepairJob*, as the *Car* is *subject of* the *RepairJob*. This relation is a one-to-one relation, as only one *Car* can be in each *RepairJob*, and it wouldn't make sense to have multiple *RepairJob*s at a time for a single *Car*. In this relation, the *RepairJob* has total participation, as a *RepairJob* has to have a car to repair. Still, the *Car* entity only has partial participation, as it could be beneficial to keep the car in the database for future repairs. Therefore, a *Car* doesn't always have to be involved in a *RepairJob*.

Requests is the relation between *RepairJob* and *Customer*. Just like the *Subject of* relation, the *Requests* relation is one-to-one, with the *RepairJob* having total participation, and the *Customer* having partial participation. This is, again, because a *RepairJob* needs a customer who requests and pays for the *RepairJob*, and it could be beneficial to save the *customer* in the database for future *RepairJob*s, meaning that a customer doesn't have to have an active *RepairJob*.

Works on is the relation between *RepairJob* and *Mechanic*. It is a many-to-many relationship, as it is both possible for multiple mechanics to work together on a single job and for a single *Mechanic* to work on multiple jobs. We also decided that both the *RepairJob* and the *Mechanic* have partial participation. This was chosen because it should be possible for the *Mechanic* to have vacation or other sorts of breaks without having any ongoing *Repair-Job*s, and it should also be possible to have a *RepairJob* that is not started yet, and should therefore not have an assigned *Mechanic* yet.

Works at is the relation between *Mechanic* and *Workshops*. We chose that the relation should be a many-to-one relation as there can be many *Mechanics* employed at a single *Workshop*. The *Mechanic* has total participation, as it does not make sense to have mechanics in our database that are not employed at any *Workshops*. The *Workshops* only have partial participation, as we thought there might be some possibility that no *Mechanics* were assigned to a given *Workshop* during renovation or for other reasons.

Happens in is the relation between *RepairJob* and *Workshop*. It is a many-to-one relation as it should be possible to have multiple *RepairJob*s at a *Workshop* at the same time, and a *RepairJob* cannot be shared across multiple *Workshops*. As with the *Mechanic* in the *Works at* relation, the *Repair-Job* has total participation, as a *RepairJob* has to happen at a location. The *Workshop* has partial participation, as it is possible for a *Workshop* to have no customers and therefore no work.

Contains is the relation between *Workshops* and *Inventory*. This model that each *Workshop* contains a number of items from an *Inventory*. As the inventory contains multiple items belonging to a single *Workshop*, we've decided that the *Contains* relation should be a many-to-one relation. The inventory has total participation as a given part has to be located somewhere, and the *Workshop* only has partial participation as it should be possible for a *Workshop* to run out of items in their inventory.

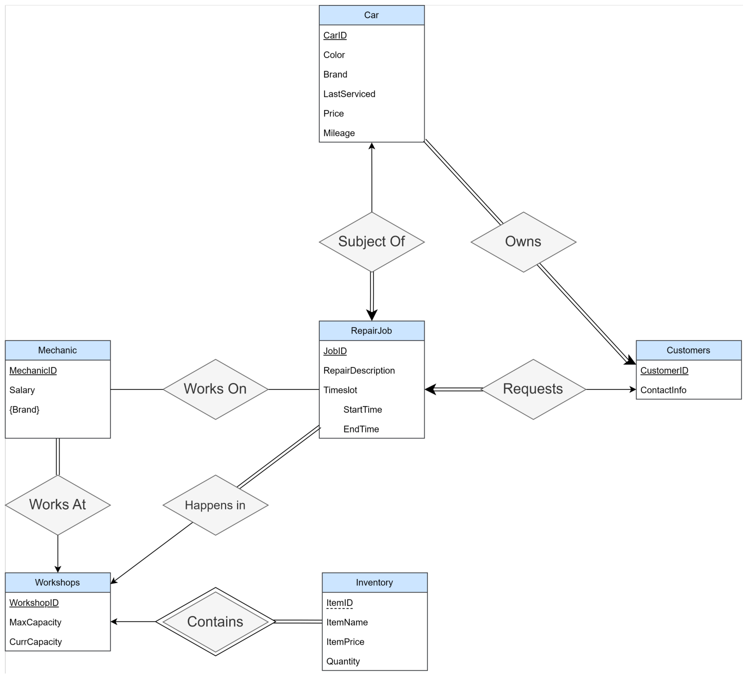


Fig. 1: Entity Relation Diagram

3 Logical Design

Most of the entity sets are quite straight forward to convert to a relation schema. We simply identified the primary keys for each relation and included those as foreign keys. However there were a few cases where a bit more work was needed. Since we want mechanics to be able to work on multiple jobs and also have multiple mechanics assigned to a single job, we had to model this relation as a set in itself. As such we made the *WorksOn* set which contains the foreign keys *MechanicID* and *JobID*. This set simply ties mechanics to jobs which lets us have a many-to-many relation between them. We also have the weak entity *Inventory* that is linked to *Workshop*. Here we added the primary key of *Workshop* to *Inventory*. Lastly we had the composite value *TimeSlot* in *RepairJob,* where we simply added the leaf values to the set and removed the root value. The result is the Relation Schema seen below and the Database diagram seen on figure 2.

```
WorksOn (MechanicID, JobID)
    Foreign key (MechanicID, JobID)
    References Mechanic(MechanicID), RepairJob(JobID)
```

Inventory(ItemID, WorkshopID, ItemName, ItemPrice, Quantity)
  Foreign key (WorkshopID)
  References Workshops(WorkshopID)

Car (CarID, CustomerID, Color, Brand, LastServiced, Price, Mileage)
  Foreign key (CustomerID)
  References Customers(CustomerID)

RepairJob (JobID, CarID, CustomerID, WorkShopID, RepairDescription, StartTime, EndTime)
  Foreign key (CarID, CustomerID, WorkshopID)
  References Car(CarID), Customers(CustomerID), Workshops(WorkShopID)

Mechanic (MechanicID, WorkshopID, Salary, Brand)
  Foreign key (WorkshopID)
  References Workshops(WorkshopID)

Customers (CustomerID, ContactInfo)

Workshops (WorkshopID, MaxCapacity, CurrCapacity)

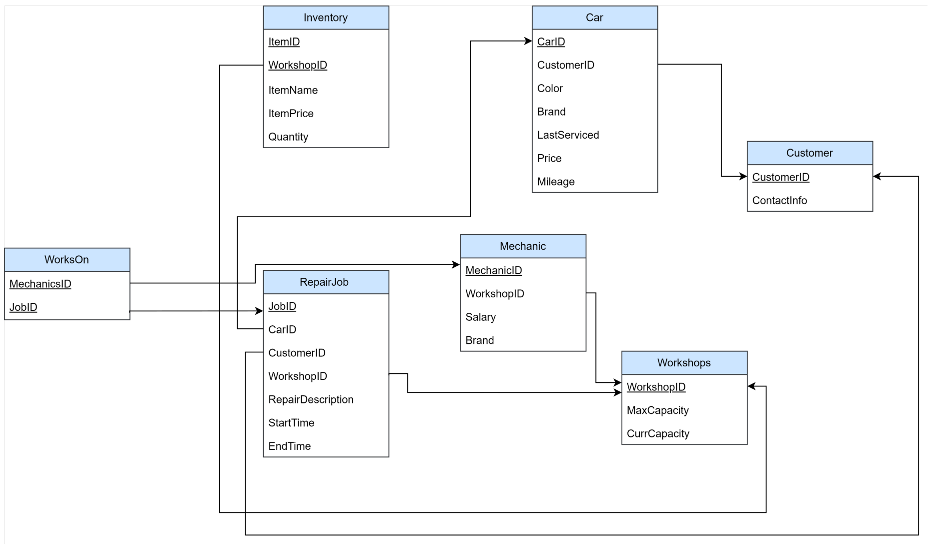
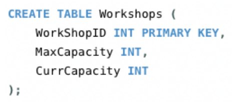


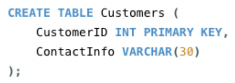
Fig. 2: DataBase Scheme Diagram

4 Implementation

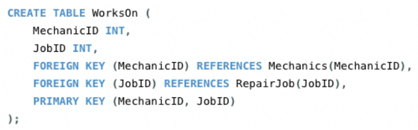
The car reparations company MariaDB database with the respective entity sets **Cars**, **Customer**, **RepairJob**, **Mechanic**, **Workshops**, **Inventory** and the relation **WorksOn** were implemented using the following SQL statements:



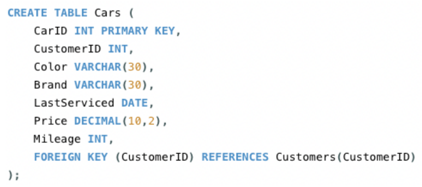
```
CREATE TABLE Workshops (
    WorkShopID INT PRIMARY KEY,
    MaxCapacity INT,
    CurrCapacity INT
);
```



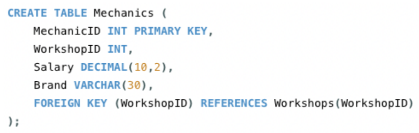
```
CREATE TABLE Customers (
    CustomerID INT PRIMARY KEY,
    ContactInfo VARCHAR(30)
);
```



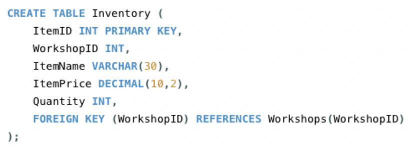
```
CREATE TABLE WorksOn (
    MechanicID INT,
    JobID INT,
    FOREIGN KEY (MechanicID) REFERENCES Mechanics(MechanicID),
    FOREIGN KEY (JobID) REFERENCES RepairJob(JobID),
    PRIMARY KEY (MechanicID, JobID)
);
```



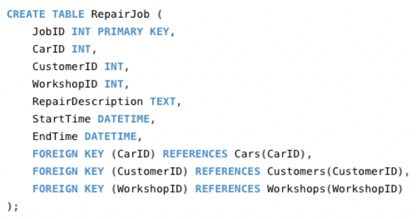
```
CREATE TABLE Cars (
    CarID INT PRIMARY KEY,
    CustomerID INT,
    Color VARCHAR(30),
    Brand VARCHAR(30),
    LastServiced DATE,
    Price DECIMAL(10,2),
    Mileage INT,
    FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)
);
```



```
CREATE TABLE Mechanics (
    MechanicID INT PRIMARY KEY,
    WorkshopID INT,
    Salary DECIMAL(10,2),
    Brand VARCHAR(30),
    FOREIGN KEY (WorkshopID) REFERENCES Workshops(WorkshopID)
);
```



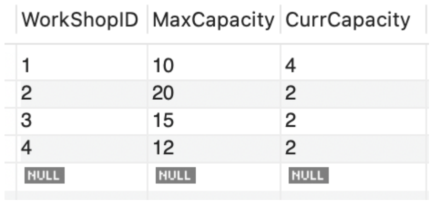
```
CREATE TABLE Inventory (
    ItemID INT PRIMARY KEY,
    WorkshopID INT,
    ItemName VARCHAR(30),
    ItemPrice DECIMAL(10,2),
    Quantity INT,
    FOREIGN KEY (WorkshopID) REFERENCES Workshops(WorkshopID)
);
```



```
CREATE TABLE RepairJob (
    JobID INT PRIMARY KEY,
    CarID INT,
    CustomerID INT,
    WorkshopID INT,
    RepairDescription TEXT,
    StartTime DATETIME,
    EndTime DATETIME,
    FOREIGN KEY (CarID) REFERENCES Cars(CarID),
    FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID),
    FOREIGN KEY (WorkshopID) REFERENCES Workshops(WorkshopID)
);
```

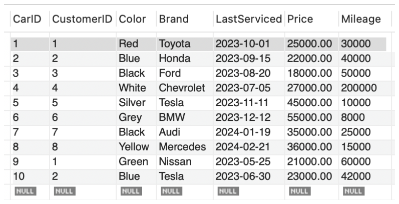
5 Database Instance

After populating each table with their respective values related to the car reparations MariaDB database, and after selecting all data rows from each table the following SQL output was achieved:



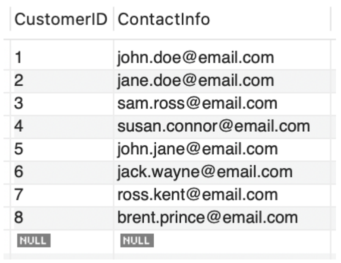
WorkShopID	MaxCapacity	CurrCapacity
1	10	4
2	20	2
3	15	2
4	12	2
NULL	NULL	NULL

(a) workshop query



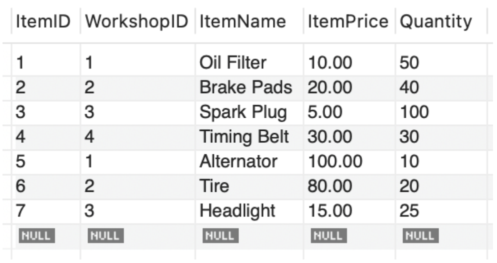
CarID	CustomerID	Color	Brand	LastServiced	Price	Mileage
1	1	Red	Toyota	2023-10-01	25000.00	30000
2	2	Blue	Honda	2023-09-15	22000.00	40000
3	3	Black	Ford	2023-08-20	18000.00	50000
4	4	White	Chevrolet	2023-07-05	27000.00	200000
5	5	Silver	Tesla	2023-11-11	45000.00	10000
6	6	Grey	BMW	2023-12-12	55000.00	8000
7	7	Black	Audi	2024-01-19	35000.00	25000
8	8	Yellow	Mercedes	2024-02-21	36000.00	15000
9	1	Green	Nissan	2023-05-25	21000.00	60000
10	2	Blue	Tesla	2023-06-30	23000.00	42000
NULL	NULL	NULL	NULL	NULL	NULL	NULL

(b) cars query



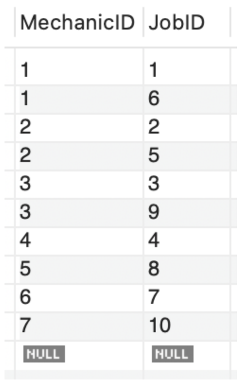
CustomerID	ContactInfo
1	john.doe@email.com
2	jane.doe@email.com
3	sam.ross@email.com
4	susan.connor@email.com
5	john.jane@email.com
6	jack.wayne@email.com
7	ross.kent@email.com
8	brent.prince@email.com
NULL	NULL

(c) customers query



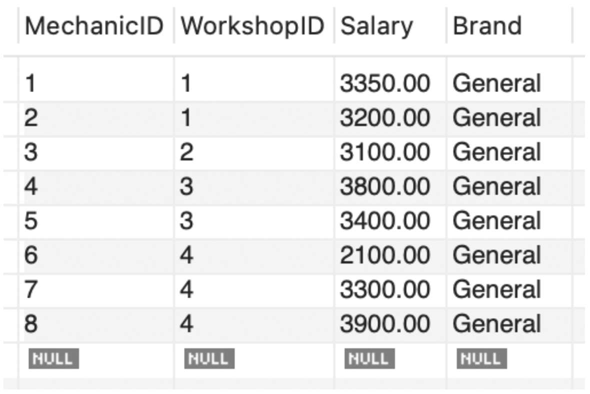
ItemID	WorkshopID	ItemName	ItemPrice	Quantity
1	1	Oil Filter	10.00	50
2	2	Brake Pads	20.00	40
3	3	Spark Plug	5.00	100
4	4	Timing Belt	30.00	30
5	1	Alternator	100.00	10
6	2	Tire	80.00	20
7	3	Headlight	15.00	25
NULL	NULL	NULL	NULL	NULL

(d) inventory query



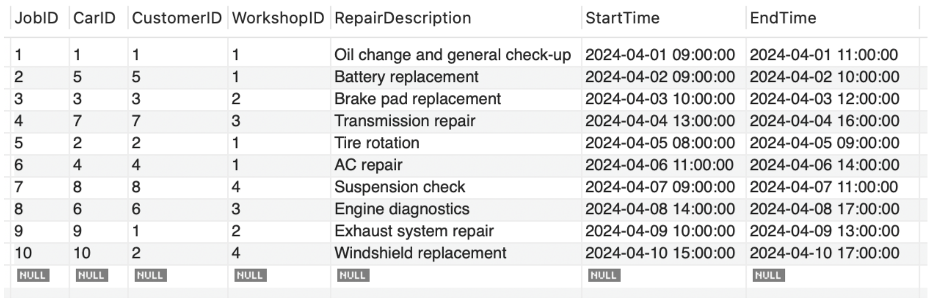
MechanicID	JobID
1	1
1	6
2	2
2	5
3	3
3	9
4	4
5	8
6	7
7	10
NULL	NULL

(e) works on query



MechanicID	WorkshopID	Salary	Brand
1	1	3350.00	General
2	1	3200.00	General
3	2	3100.00	General
4	3	3800.00	General
5	3	3400.00	General
6	4	2100.00	General
7	4	3300.00	General
8	4	3900.00	General
NULL	NULL	NULL	NULL

(f) mechanics query



JobID	CarID	CustomerID	WorkshopID	RepairDescription	StartTime	EndTime
1	1	1	1	Oil change and general check-up	2024-04-01 09:00:00	2024-04-01 11:00:00
2	5	5	1	Battery replacement	2024-04-02 09:00:00	2024-04-02 10:00:00
3	3	3	2	Brake pad replacement	2024-04-03 10:00:00	2024-04-03 12:00:00
4	7	7	3	Transmission repair	2024-04-04 13:00:00	2024-04-04 16:00:00
5	2	2	1	Tire rotation	2024-04-05 08:00:00	2024-04-05 09:00:00
6	4	4	1	AC repair	2024-04-06 11:00:00	2024-04-06 14:00:00
7	8	8	4	Suspension check	2024-04-07 09:00:00	2024-04-07 11:00:00
8	6	6	3	Engine diagnostics	2024-04-08 14:00:00	2024-04-08 17:00:00
9	9	1	2	Exhaust system repair	2024-04-09 10:00:00	2024-04-09 13:00:00
10	10	2	4	Windshield replacement	2024-04-10 15:00:00	2024-04-10 17:00:00
NULL	NULL	NULL	NULL	NULL	NULL	NULL

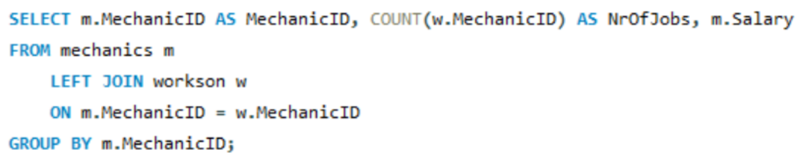
(g) repairjob query

6 SQL Data Queries

To demonstrate how the database can be used to extract specific information, we will provide three examples of typical SQL query statements using joins, group by, and set operations.

Example 1

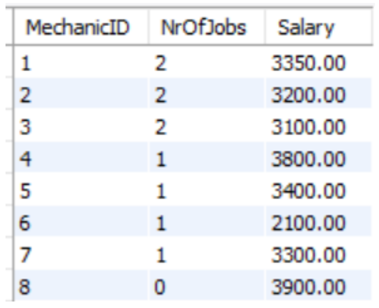
For the first example, SQL query statements using joins and group by were employed. The following SQL query statement was used:



```
SELECT m.MechanicID AS MechanicID, COUNT(w.MechanicID) AS NrOfJobs, m.Salary
FROM mechanics m
    LEFT JOIN workson w
    ON m.MechanicID = w.MechanicID
GROUP BY m.MechanicID;
```

Fig. 5: Query example 1 using JOIN and GROUP BY

The above SQL statement Fig. 5 query compares all the mechanics, showing their salaries and shows how many jobs they are currently working on. The query LEFT JOINs the Mechanics table with the WorksOn table on the MechanicID, so that it shows all mechanics, even if a mechanic isn't represented in the WorksOn table(meaning he doesn't currently have a job to work on). It then groups the results of the join query by MechanicID to make sure the count of NrOfJobs is shown separately for each mechanic. The results from the query are given below in Fig. 6.

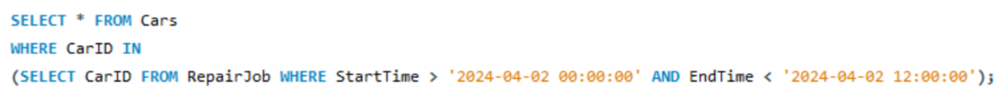


MechanicID	NrOfJobs	Salary
1	2	3350.00
2	2	3200.00
3	2	3100.00
4	1	3800.00
5	1	3400.00
6	1	2100.00
7	1	3300.00
8	0	3900.00

Fig. 6: Output from query example 1 using JOIN and GROUP BY

Example 2

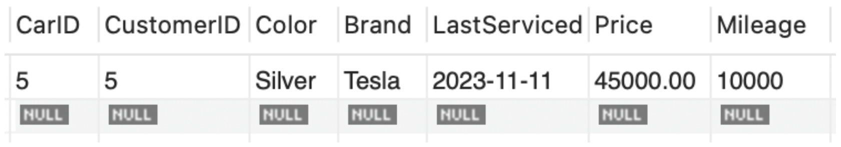
For the second example, SQL query statements using the IN set operation was used. The following SQL query statement was used:



```
SELECT * FROM Cars
WHERE CarID IN
(SELECT CarID FROM RepairJob WHERE StartTime > '2024-04-02 00:00:00' AND EndTime < '2024-04-02 12:00:00');
```

Fig. 7: Query example 2 using the IN set operation

The above SQL Fig. 7 query asks for all the details of cars in the car repair shop, that had a repair job that started 2. April 2024 and ended the same day. The query uses a sub-query to find all the CarIDs with a start time after the specified date and an end time after the specified date. It then selects all the details from Cars for the selected CarIDs.

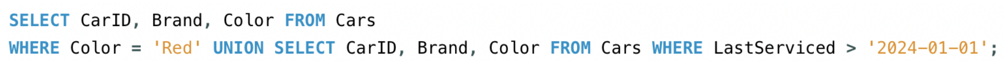


CarID	CustomerID	Color	Brand	LastServiced	Price	Mileage
5	5	Silver	Tesla	2023-11-11	45000.00	10000
NULL	NULL	NULL	NULL	NULL	NULL	NULL

Fig. 8: Output from query example 2 using the IN set operation

Example 3

The final example of a SQL query statement utilizes the UNION set operation, as shown in the SQL query statement in Fig. 9.

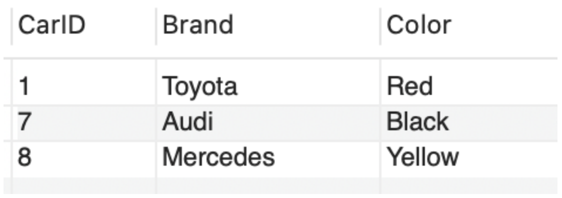


```
SELECT CarID, Brand, Color FROM Cars
WHERE Color = 'Red' UNION SELECT CarID, Brand, Color FROM Cars WHERE LastServiced > '2024-01-01';
```

Fig. 9: Query example 3 using the UNION set operation

The SQL query in Fig. 9 asks for all of the cars that are either red or have been serviced after a particular date, namely 1. January 2024. It utilizes the UNION operator to merge two sets of results: one set containing cars that are red, and another set consisting of cars that were last serviced after 1.

January 2024. It is important to note that the UNION operation removes any duplicate entries, hence, the result is a table with the CarIDs, colors, and brands of cars that are either red or were last serviced after the specific date.



CarID	Brand	Color
1	Toyota	Red
7	Audi	Black
8	Mercedes	Yellow

Fig. 10: Output from query example 2 using the IN set operation

7 SQL Programming

In this section, we show a `TRIGGER`, a `PROCEDURE`, and a `FUNCTION` to help us maintain our database.

TRIGGER

We've made a trigger called *SelectSuitableMechanic*. Its main function is to automatically assign RepairJobs to Mechanics as the repairJobs are inserted, additionally it keeps track of the CurrCapacity of the corresponding workshop, so that a new repairJob cannot be added if currCapacity will exceed maxCapacity.

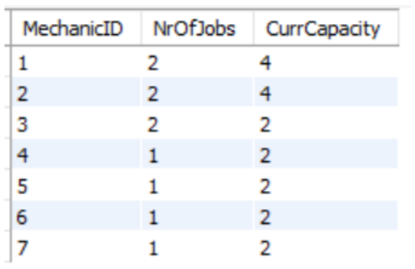
The query contains 3 parts, 1) Find a suitable mechanic for the job, 2) validity-checks, 3) execute changes.

When finding a suitable mechanic, there are a few criteria that we account for. First, we need the mechanic to be able to work with the brand of the car(mechanic and car need to have the same value in the field *Brand*). secondly, we need the mechanic to work at the same workshop as the RepairJob has been created(mechanic.Workshop needs to be the same as RepairJob.Workshop). The last criteria is that we automatically select the suitable mechanic with the fewest amount of current RepairJobs.

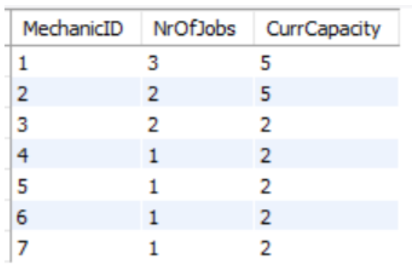
If the combination of the two first criteria excludes all the mechanics, the chosen mechanic will be `NULL`, which we can use in the validity-check step. In this step, we raise a signal if the mechanic is null or if $Workshop.currCapacity >= Workshop.maxCapacity$.

In the last step, we increment the Workshop.currCapacity and inserts a row into the *WorksOn*-table stating that the chosen mechanic is now working on the given job.

To illustrate the effect of the trigger, we inserted a row in RepairJob. The relevant columns can be seen in Fig 11. Both the `INSERT`-statement and the `SELECT`-statement is included at the bottom of the *10_02170DatabaseScript2_2024* file under "Miscellaneous"



MechanicID	NrOfJobs	CurrCapacity
1	2	4
2	2	4
3	2	2
4	1	2
5	1	2
6	1	2
7	1	2



MechanicID	NrOfJobs	CurrCapacity
1	3	5
2	2	5
3	2	2
4	1	2
5	1	2
6	1	2
7	1	2

Fig. 11: The output before(left) and after(right) INSERTing a row in RepairJob and triggering the TRIGGER

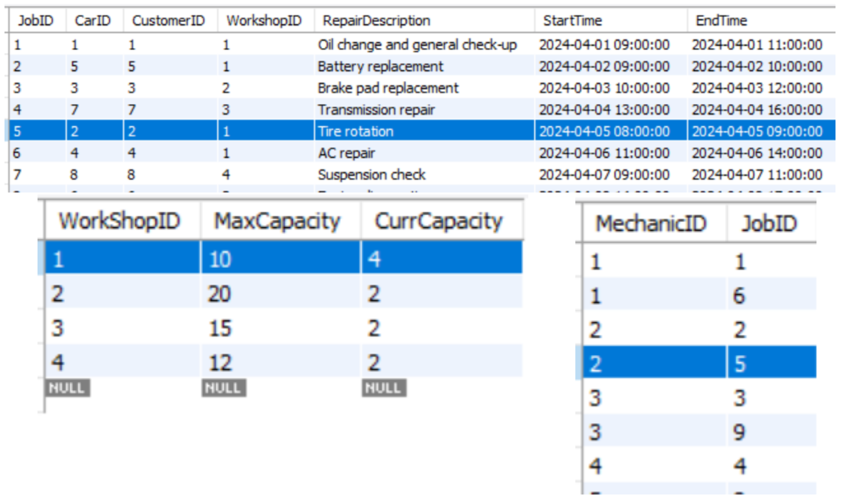
PROCEDURE

The Procedure we made is called *DeleteRepairJob*, and as the name states, it's used for deleting repairJobs and the necessary data related to the repairJob. The repairJob's primary key is used in the table *WorksOn* and thus we have to delete the relevant rows here first, additionally, we have to decrement the current capacity of the given workshop so that it continues to match the number of jobs in the given workshop. We chose not to delete the car and the customer related to the repairJob, as it might be relevant to save them for future repairs.

To illustrate the effects of the *DeleteRepairJob* procedure, we play the scenario of a job being completed. We chose to delete the repairJob with $JobID = 5$, the tables before executing the procedure can be seen in Fig 12, and the tables after executing the procedure can be seen in Fig 13.

FUNCTION

The function we made is called *NumOfJobs*. It takes a MechanicID and returns the number of jobs that the mechanic is currently working on. It's a simple function that counts the number of rows in *WorkingOn* with the given MechanicID. The same result can be gained by using joins, but this function makes it easier to work with. An example output of this can be seen in Fig. 14.

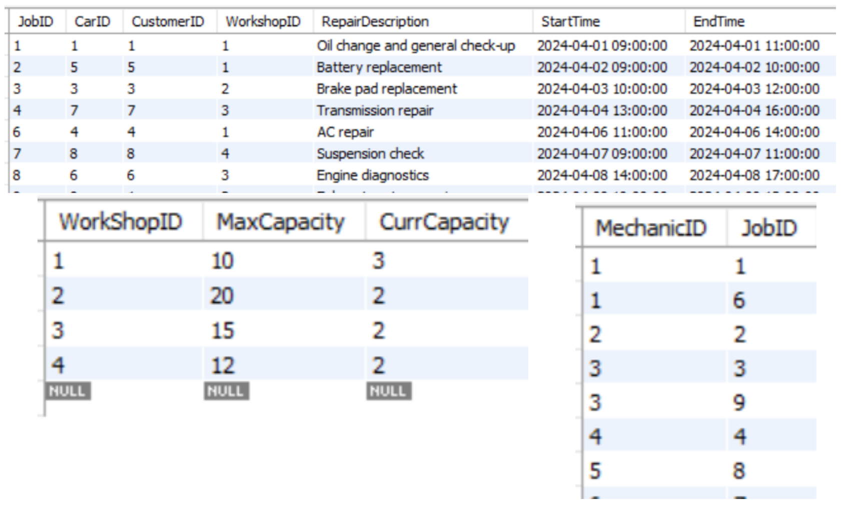


JobID	CarID	CustomerID	WorkshopID	RepairDescription	StartTime	EndTime
1	1	1	1	Oil change and general check-up	2024-04-01 09:00:00	2024-04-01 11:00:00
2	5	5	1	Battery replacement	2024-04-02 09:00:00	2024-04-02 10:00:00
3	3	3	2	Brake pad replacement	2024-04-03 10:00:00	2024-04-03 12:00:00
4	7	7	3	Transmission repair	2024-04-04 13:00:00	2024-04-04 16:00:00
5	2	2	1	Tire rotation	2024-04-05 08:00:00	2024-04-05 09:00:00
6	4	4	1	AC repair	2024-04-06 11:00:00	2024-04-06 14:00:00
7	8	8	4	Suspension check	2024-04-07 09:00:00	2024-04-07 11:00:00

WorkShopID	MaxCapacity	CurrCapacity
1	10	4
2	20	2
3	15	2
4	12	2
NULL	NULL	NULL

MechanicID	JobID
1	1
1	6
2	2
2	5
3	3
3	9
4	4

Fig. 12: The output of the three affected tables before executing the procedure

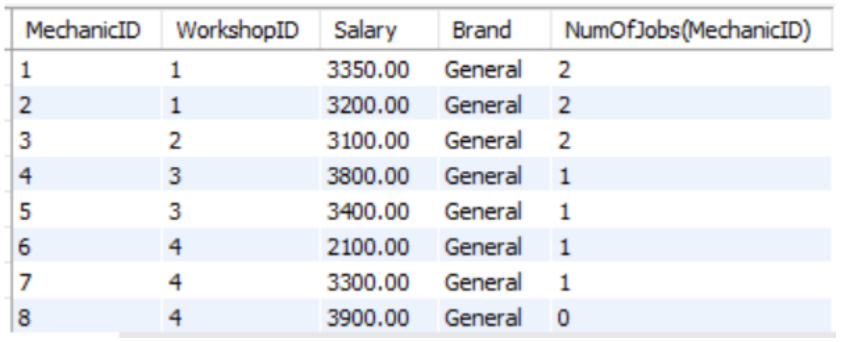


JobID	CarID	CustomerID	WorkshopID	RepairDescription	StartTime	EndTime
1	1	1	1	Oil change and general check-up	2024-04-01 09:00:00	2024-04-01 11:00:00
2	5	5	1	Battery replacement	2024-04-02 09:00:00	2024-04-02 10:00:00
3	3	3	2	Brake pad replacement	2024-04-03 10:00:00	2024-04-03 12:00:00
4	7	7	3	Transmission repair	2024-04-04 13:00:00	2024-04-04 16:00:00
6	4	4	1	AC repair	2024-04-06 11:00:00	2024-04-06 14:00:00
7	8	8	4	Suspension check	2024-04-07 09:00:00	2024-04-07 11:00:00
8	6	6	3	Engine diagnostics	2024-04-08 14:00:00	2024-04-08 17:00:00

WorkShopID	MaxCapacity	CurrCapacity
1	10	3
2	20	2
3	15	2
4	12	2
NULL	NULL	NULL

MechanicID	JobID
1	1
1	6
2	2
3	3
3	9
4	4
5	8

Fig. 13: The output of the three affected tables after executing the procedure

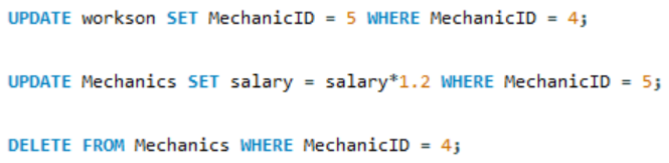


MechanicID	WorkshopID	Salary	Brand	NumOfJobs(MechanicID)
1	1	3350.00	General	2
2	1	3200.00	General	2
3	2	3100.00	General	2
4	3	3800.00	General	1
5	3	3400.00	General	1
6	4	2100.00	General	1
7	4	3300.00	General	1
8	4	3900.00	General	0

Fig. 14: Query example 1 using JOIN and GROUP BY

8 SQL Table Modifications

In this section we demonstrate some examples of how we can modify our database using `UPDATE` and `DELETE` statements. The scenario we chose to show is a mechanic getting fired. There are three parts to our scenario, transferring RepairJobs from the mechanic being fired to another mechanic, raising the salary of the mechanic getting the extra jobs, and then deleting the mechanic getting fired. The Queries used for the scenario are shown in Fig 15.



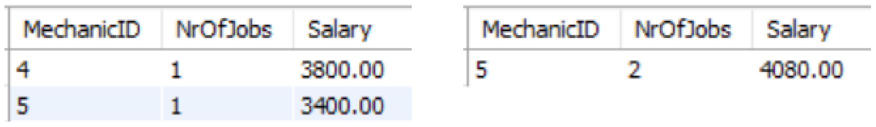
```
UPDATE workson SET MechanicID = 5 WHERE MechanicID = 4;

UPDATE Mechanics SET salary = salary*1.2 WHERE MechanicID = 5;

DELETE FROM Mechanics WHERE MechanicID = 4;
```

Fig. 15: The three queries

In our scenario, we chose that the mechanic with MechanicID = 4 is the mechanic that gets fired, and the mechanic with MechanicID = 5 is getting the extra tasks and a 20% increase to his salary. The outputs are shown in Fig 16. (The script for showing the output is included at the bottom of the *10_02170DatabaseScript2_2024* file under "Miscellaneous")



MechanicID	NrOfJobs	Salary
4	1	3800.00
5	1	3400.00

MechanicID	NrOfJobs	Salary
5	2	4080.00

Fig. 16: The output before(left) and after(right) executing the scenario