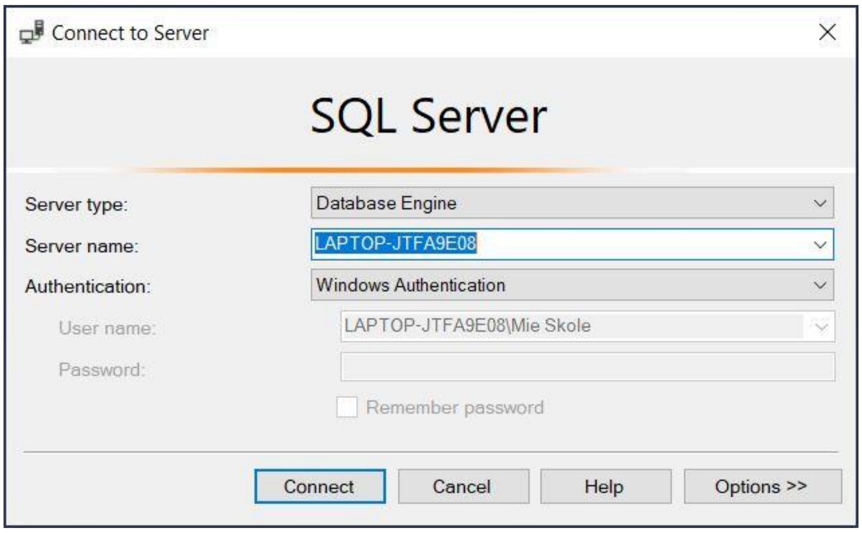
Car - Python DB Handin

Web Developer 1st Semester

Mie Grønkjær Bechmann Jørgensen
cph-mj761@cphbusiness.dk

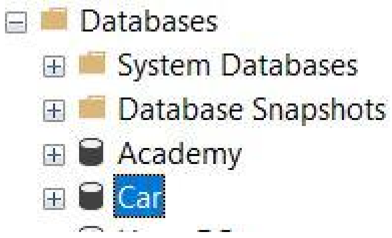
Database

The Database



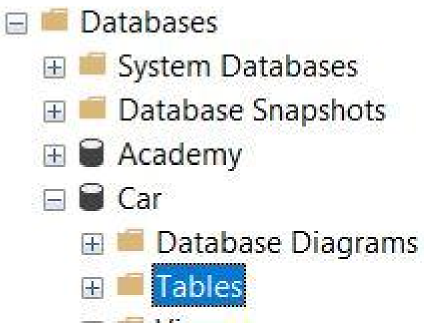
Server type: Database Engine
Server name: LAPTOP-JTFA9E08
Authentication: Windows Authentication
User name: LAPTOP-JTFA9E08\Mie Skole
Password:
Remember password

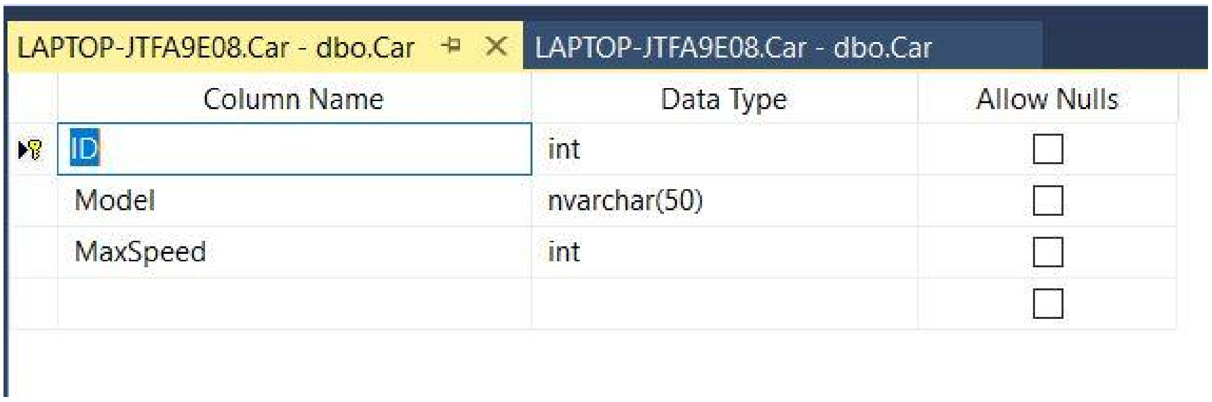
I start off by creating a new database inside of SQL Server, called Car.



Table

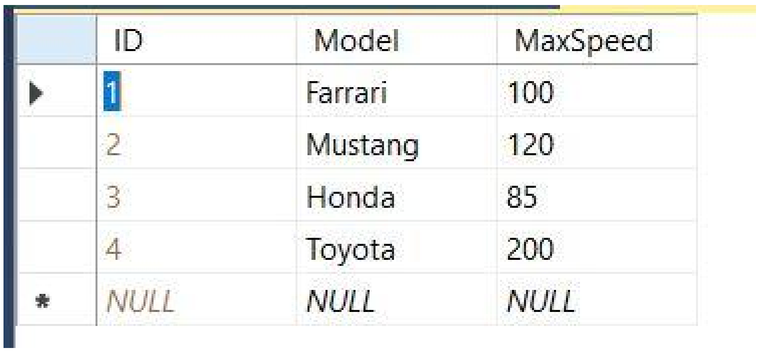
Inside the database I then create a new table which I also call Car, just to keep things simple.
After that I put in the information inside the table. Just like the assignment I give it an ID and make sure that's the primary key. As well as a Model and a MaxSpeed, Where I give ID and MaxSpeed with the data types int. And the Model as an nvarchar, and all of them does not allow null since we would like all the information to be input when creating a new car.
As well as remember to go into, is identity, and change it to yes to create an auto ID generator.




LAPTOP-JTFA9E08.Car - dbo.Car | LAPTOP-JTFA9E08.Car - dbo.Car

Column Name	Data Type	Allow Nulls
ID	int	☐
Model	nvarchar(50)	☐
MaxSpeed	int	☐
		☐

I then pre-create a few cars inside the edit window, to have a few cars inside the database to work with in the beginning.



ID	Model	MaxSpeed
1	Farrari	100
2	Mustang	120
3	Honda	85
4	Toyota	200
NULL	NULL	NULL

Visual Studio

Python file

I then open up Microsoft Visual Studio and start by creating a new Python web project since we do wanna create a webpage.
I call the file CarHandin, and then proceed to create a new empty python file and call it MyApplication. This is gonna be the page where I put in all my CRUD code and is gonna be the only python file I have.

Templates Folder

I created a new folder called templates, this is where I am gonna have my webpages. Inside the templates folder I create two web files, Index.html and edit.html.

Code - SQL Server Connection

Important files I need to import or get in order for it to work are: Flask (for the web connection, as well as render template), pyodbc (installed by pip and connects SQL server), as well as os.
I will need all of these to connect everything.

When creating the code conn =, which we have used in other examples it is important to remember to input your pc server in order for it to work.

```
from flask import Flask, render_template, request
import pyodbc # installed by pip
import os
# Flask looks at the folder templates and thats just how that works.

app = Flask(__name__)

conn = pyodbc.connect(Trusted_Connection = 'yes', driver = '{SQL
Server}', server = 'LAPTOP-JTFA9E08', database = 'Car')
cursor = conn.cursor()

@app.route('/')
@app.route('/index')
def showindexpage() :
    cursor.execute("select * from Car")
    mydata = cursor.fetchall()
    #conn.close()
```

This connects everything to the index file I have created using @app.route(). Then I tell the program that I want the cursor (cursor allows SQL code to be executed in the python file), to execute every data from the database Car, Since we wanna use all the data on our page. Inside this code I wanna show the fastest car using the already input cars from the database, which I define by using:

```
 #max speed
    cursor.execute("select * from Car order by MaxSpeed DESC")
    fastest_car = cursor.fetchone() #one row - first

    return render_template("index.html", values = mydata,
                           car = fastest_car)
```

I tell the program I want it in order of the fastest car and tells the program to return inside the rendered index rendered template. It only shows the fastest car since I told the program to only show one row and the fastest car will be the top one since we ordered it by MaxSpeed.

Create

I used Flasks route to create a parameter called /create, for creating a new car. I also use the POST method which allows the program to send data to the server I have when creating or updating information in the database.
I start off by making a validator first. Telling the program if both values are there then it will allow a new car to be created.

```
@app.route('/create', methods=['POST'])
```

```
#validator
def create():
   if request.form['Model'] == '' or request.form['MaxSpeed'] == '':
       return showindexpage()
```

Telling the program to find the forms from the index file where I use the ‘Model’ and ‘MaxSpeed’ values inside, to connect the code to the html. It tells the program that when these values are input this new data should be sent to the Car database inside og the Model and MaxSpeed values. And in order for it to work I remember to end the code with cursor.commit to commit the creation.

```
#create
   cursor.execute("INSERT into Car VALUES (?, ?)",
request.form['Model'], request.form['MaxSpeed'])
   cursor.commit()
   return showindexpage()
```

Edit - ID get

I use this code to catch the auto ID I created when making my table.

```
#edit
@app.route('/search', methods=['GET'])
def search():
    id = request.args.get('id')
    cursor.execute("SELECT * from Car WHERE ID = ?", id)
    mydata = cursor.fetchone()
    return render_template('edit.html', values = mydata)
```

As you can see it is very similar to some of the other code bits created. It just tells the program where to GET the data I need from my database using a ? since it is auto generated. As well as telling the program where I use this code in the edit.html page.

Update

I again use Flasks route and create a parameter update again with the POST method. Telling the program that if the forms Model and MaxSpeed are requested to change, shown by the empy ‘ ‘ (strings) it should return the index page. As well as get all the information Model and Speed, since these are the values that the user can change.
I tell the program inside the Cursor.execute (which allows the python file to write and execute the SQL text), that all of these values can be changed by the user ‘?’. And that the program should get the ID number from the code we just created, since we do not allow the ID to be updated. Again ending with a commit.

```
#update
@app.route('/update', methods=['POST'])
def update():
    if request.form['Model'] == '' or request.form['MaxSpeed'] == '':
      return showindexpage()
    id = int(request.form['ID'])
    model = request.form['Model']
    max_speed = float(request.form['MaxSpeed'])
    cursor.execute("UPDATE Car SET Model = ?, MaxSpeed = ? WHERE ID =
?", model,max_speed,id)
    cursor.commit()
```

```
return showindexpage()
```

Delete

I Again use the Flask route and the parameter delete for the delete part. Telling the program to GET the information from the database which needs to be deleted.
With the SQL code I use the method DELETE to tell it when the button delete is pressed it should get all the data from that file and delete that from the database.

```
#delete
@app.route('/delete', methods=['GET'])
def delete():
    id = request.args.get('id')
    cursor.execute("DELETE from Car WHERE ID = ?", id)
    cursor.commit()
    return showindexpage()
```

Finishing the python code off I tell it to execute the html page in the localhost page 4449.

```
if __name__ == '__main__' :
    app.run('localhost', 4449)
# set port on properties = Dubug = portnumber
```

HTML Pages

Index.html

I am using both Bootstrap for the look of the page as well as Jinja2.
I start off my design by putting everything inside a div with the bootstrap container class. As well as a div with the row class. This will make my setup much more visually pleasing for the eye.

Inside this I first create a H2 tag to show the fastest car.
I then use a div where I put in the data using Jinja2 and text. Inputting the data I have into the text area. Since we already have defined that we want the cars ordered by Maxspeed, the fastest car will be shown first and will first only show one row.

```
<div class="container">
        <div class="row">
            <div class="col-6">
                <h2>Data from Car Database</h2>
                <br />
                <div>
                    Fastest Car: {{ car.Model }}
                    <br />
                    Max Speed: {{ car.MaxSpeed }}
                </div>
```

I then create a table with table rows and table headercells (the titles of the data). And create another one where the data values are input using Jinja2. Since they work like arrays the first car would be 0 and the third car would be 2, so we input those values inside the Jinja code. I also implement an edit and delete button to all the td's with a link towards the edit and delete code we already have created in the python file.

```
</thead>
          <tbody>
            {% for row in values %} <!--for loop in Jinja-->
            <tr>
              <td>{{ row[0] }}&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;</td>
              <td>{{ row[1] }}&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;</td>
              <td>{{ row[2] }}&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;</td>
              <td>
                <a class="btn btn-info" href="/search?id={{ row[0] }}">Edit</a>
                <a class="btn btn-danger" href="/delete?id={{ row[0] }}">Delete</a>
              </td>
            </tr>
            {% endfor %}
          </tbody>
        </table>
      </div>
```

The next section is for creating or inserting a new car into the database.
It is put together using the /create parameter and the POST method since it is sending new data into the database, and I have already told the program what happens when create is executed.
It get's a title as well, and the two forms which are connected to the create code from earlier, so the program knows that these values should have input in order to create a new car.
Along with a create button which is also connected to the code.

```
 <!--Insert-->
            <div class="col">
```

```
                <form action="/create" method="post">
                    <h2>Create car</h2>
                    <div class="mb-3">
                        <label for="Model"
class="form-label">Model</label>
                        <input type="text" class="form-control"
id="Model" name="Model">
                    </div>
                    <div class="mb-3">
                        <label for="MaxSpeed" class="form-label">Max
Speed</label>
                        <input type="number" step="0.01"
class="form-control" id="MaxSpeed" name="MaxSpeed">
                    </div>
                    <div class="mb-3">
                        <button type="submit" class="btn btn-success
mb-3">Create</button>
                    </div>
                </form>
```

Edit.html

Using the same method as before on the index.html file with create, I use the /update parameter and POST method. This connects the html code to the edit and update python code.
I then set it all up using forms, labels and buttons. I use Jinja2 to input the database values in the correct form inputs. Since this is connected to the edit and update code the page knows which values can be updated.

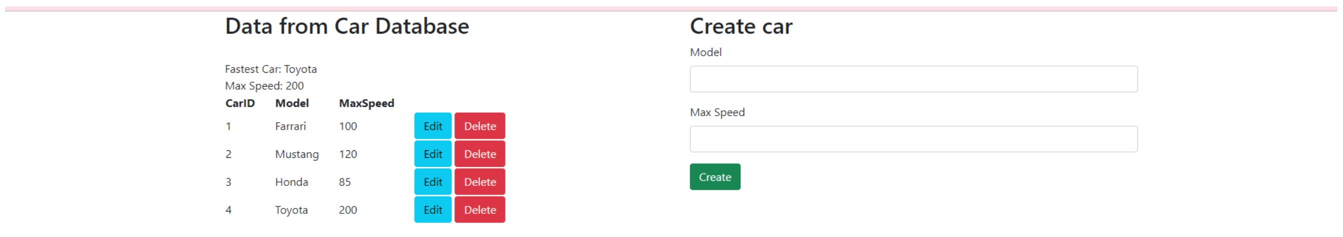
```
    <main class="container my-5">
        <form action="/update" method="post">
            <h1>Edit car database</h1>

            <div class="mb-3">
                <label for="ID" class="form-label">ID</label>
                <input type="number" readonly class="form-control"
id="ID" value="{{ values.ID }}" name="ID">
            </div>

            <div class="mb-3">
                <label for="Model" class="form-label">Model</label>
                <input type="text" class="form-control" id="Model"
value="{{ values.Model }}" name="Model">
```

```
            </div>

            <div class="mb-3">
                <label for="MaxSpeed" class="form-label">Max
Speed</label>
                <input type="number" step="0.01" class="form-control"
id="MaxSpeed" value="{{ values.MaxSpeed }}" name="MaxSpeed">
            </div>

            <div class="mb-3">
                <button type="submit" class="btn btn-primary
mb-3">Update</button>
                <a href="./" class="btn btn-danger mb-3">Cancel</a>
            </div>
        </form>

    </main>
```

Design

My design is just the default looks of the bootstrap code, it does have some different codes for colors, which I used on the buttons like danger and info, which gives the buttons a read and teal color.
I tried using a style input into the html however the code started to not work when doing so, which made me just keep the default mode.

Front page



Data from Car Database

Fastest Car: Toyota
Max Speed: 200

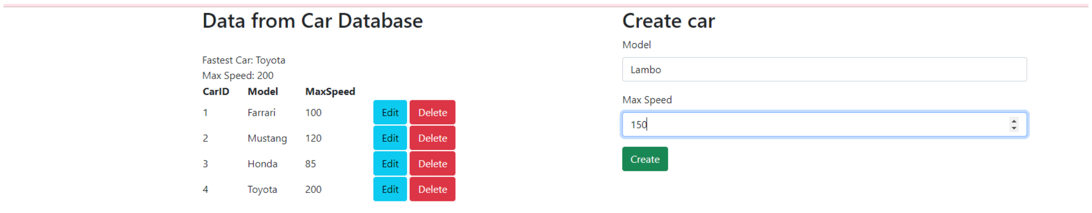
CarID	Model	MaxSpeed		
1	Farrari	100	Edit	Delete
2	Mustang	120	Edit	Delete
3	Honda	85	Edit	Delete
4	Toyota	200	Edit	Delete

Create car

Model

Max Speed

New Car



Data from Car Database

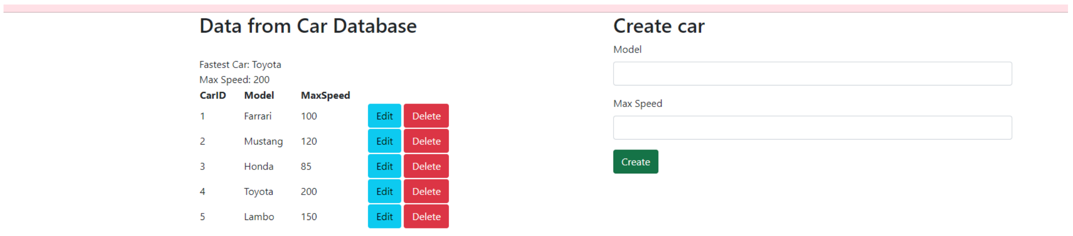
Fastest Car: Toyota
Max Speed: 200

CarID	Model	MaxSpeed		
1	Farrari	100	Edit	Delete
2	Mustang	120	Edit	Delete
3	Honda	85	Edit	Delete
4	Toyota	200	Edit	Delete

Create car

Model	Lambo
Max Speed	150

Create



Data from Car Database

Fastest Car: Toyota
Max Speed: 200

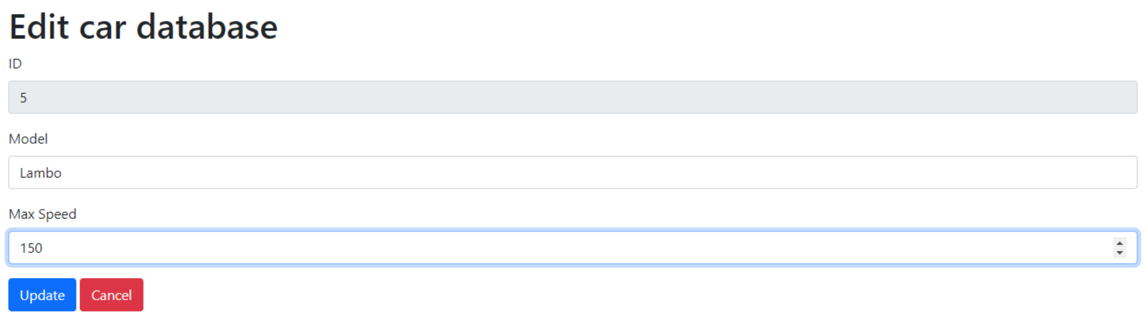
CarID	Model	MaxSpeed		
1	Farrari	100	Edit	Delete
2	Mustang	120	Edit	Delete
3	Honda	85	Edit	Delete
4	Toyota	200	Edit	Delete
5	Lambo	150	Edit	Delete

Create car

Model	
Max Speed	

Create

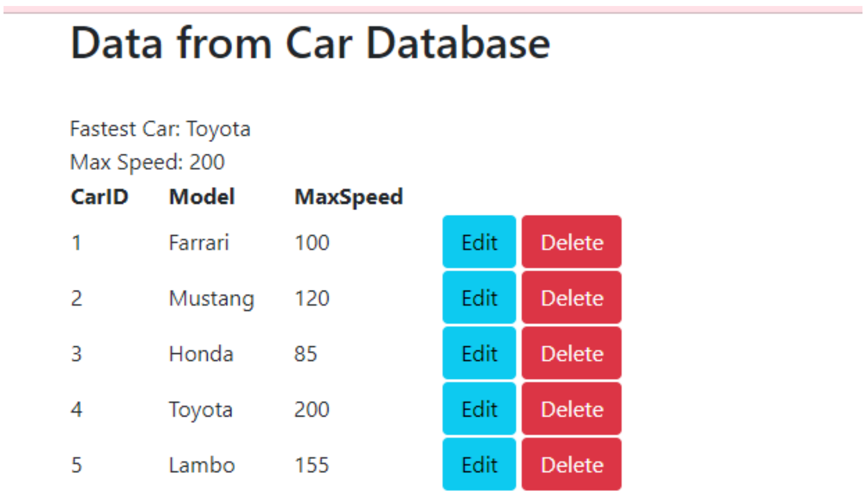
Edit car



Edit car database

ID	5
Model	Lambo
Max Speed	150

Update Cancel

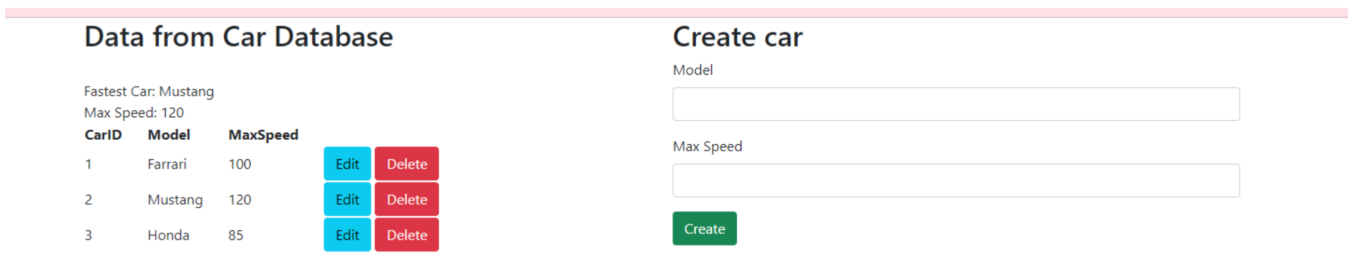


Data from Car Database

Fastest Car: Toyota
Max Speed: 200

CarID	Model	MaxSpeed		
1	Farrari	100	Edit	Delete
2	Mustang	120	Edit	Delete
3	Honda	85	Edit	Delete
4	Toyota	200	Edit	Delete
5	Lambo	155	Edit	Delete

Delete



Data from Car Database

Fastest Car: Mustang
Max Speed: 120

CarID	Model	MaxSpeed	
1	Farrari	100	Edit Delete
2	Mustang	120	Edit Delete
3	Honda	85	Edit Delete

Create car

Model

Max Speed