

# Probeklausur für POS: Fahrzeugvermietung

## Umgebung

Im Labor steht Visual Studio 2017 mit der .NET Version 2.1.520 zur Verfügung. Die C# Sprachversion ist 7.3. Daher können keine nullable reference Types oder records verwendet werden. Alle erstellten C# Projektdateien (csproj) müssen sich daher auf .NET Core 2.1 beziehen:

```
<PropertyGroup>
  <TargetFramework>netcoreapp2.1</TargetFramework>
</PropertyGroup>
```

## Musterprojekt

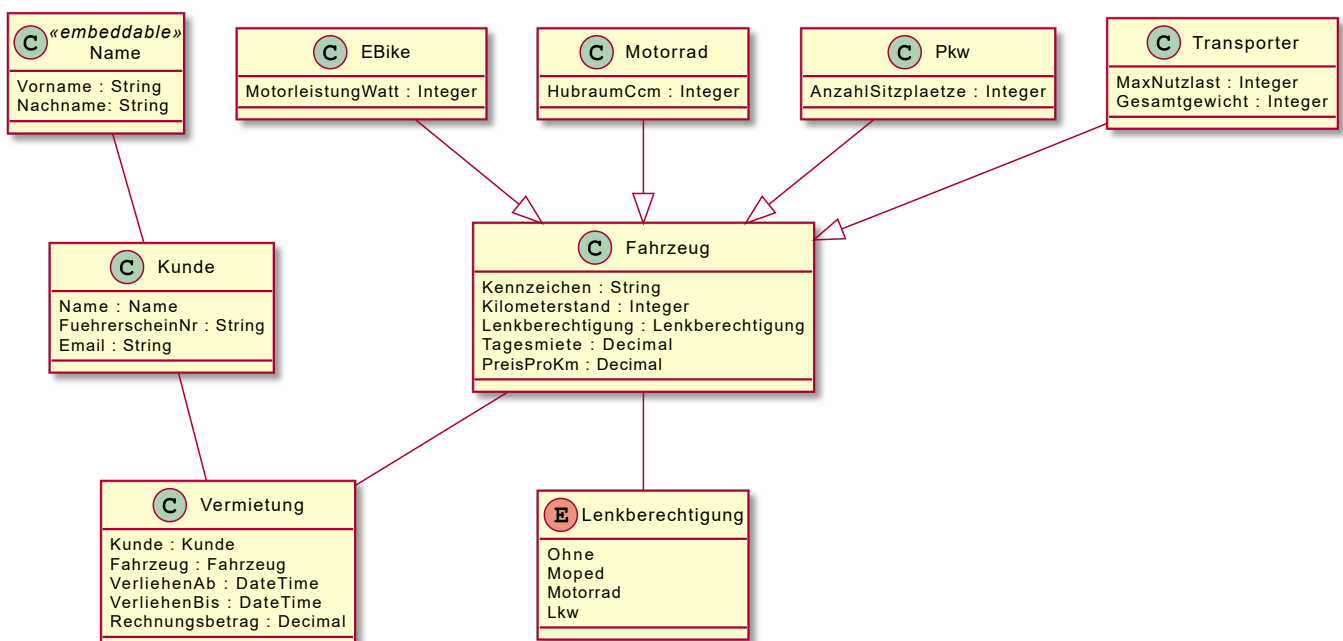
### Über die Aufgabenstellung

Im Rahmen dieser Probeklausur soll das Backend einer kleinen Fahrzeugvermietung implementiert werden. Es werden Fahrzeuge verschiedener Kategorien (PKW, Transporter, Motorrad und neuerdings auch eBikes) an Kunden vermietet. Da die eBikes bis zu 45 km/h schnell fahren, werden sie in auch als Fahrzeuge behandelt.

Die erforderliche *Lenkberechtigung* wird nur gespeichert, wenn sie von einem Autoführerschein (Kategorie B) abweicht. Das ist bei stärkeren Motorrädern oder Transportern über 3.5t der Fall. eBikes fallen bis 250W unter die Kategorie "ohne", darüber ist ein Mopedausweis erforderlich.

Die Miete richtet sich nach gefahrenen Kilometern sowie einen pro Fahrzeugtyp definierten (täglichen) Grundpreis. Der Preis pro Kilometer ist natürlich abhängig vom konkreten Fahrzeug, für das Benutzen eines Tesla ist mehr zu bezahlen als für einen klassischen Kleinwagen.

Das Domainmodel wurde aufgrund von Vorbesprechungen schon entwickelt und hat folgendes Aussehen:

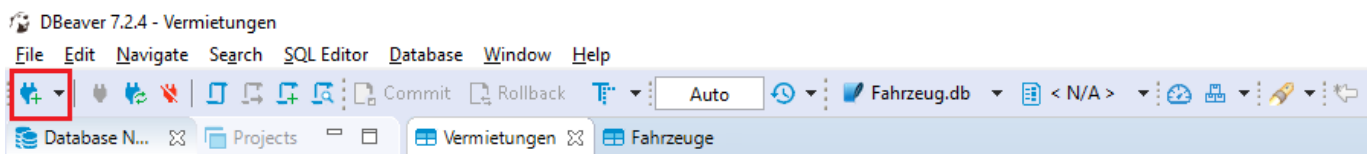


## Teilthema Domain Model, O/R Mapping und Persistence

Ihre Aufgabe ist es, das entworfene UML Klassendiagramm in C# Klassen umzusetzen. Sie können dafür einfache POCO Klassen mit offenen set Properties verwenden. Records aus C# 9 oder Klassen mit Konstruktoren können durch die vorgegebene EF Core 2 Umgebung nicht verwendet werden.

Die erstellten Klassen sollen in eine physische SQLite Datenbank persistiert werden. Anforderungen hinsichtlich der Primärschlüssel (neue Id Felder oder bestehende Felder), der konkreten Datentypen (nullable, Länge, ...) sind keine vorgeschrieben. Es muss jedoch der in der Datei [data.sql](#) vorhandene SQL Dump eingespielt werden können.

Sie können zum Einspielen des SQL Dumps in die generierte SQLite Datenbank den Datenbankeditor [Dbeaver](#) verwenden. Danach verbinden Sie sich über das Connect Symbol zur Datenbank:



Nachdem die Datei [data.sql](#) importiert wurde, implementieren Sie ein *CarRepository*, welches folgende Abfragen erlaubt:

- Abfrage *eines Fahrzeuges* auf Basis des Kennzeichens.
- Abfrage *von Fahrzeugen*, die mit einer bestimmten Lenkberechtigung gefahren werden dürfen.
- Abfrage *von Fahrzeugen*, dessen Kennzeichen mit einem bestimmten Bundesland beginnt (W... für Wien).

Sie können die Funktionen auch im DbContext implementieren, welche das Ergebnis als *IQueryable* zurückgeben.

Schreiben Sie Tests in xUnit, welches diese Funktionen aufgrund der Testdaten überprüft.

## Teilthema Service Layer / Business Logic


Schreiben Sie eine Klasse *RentalService*, die zwei für den Businessbetrieb erforderlichen Methoden beinhaltet:

**TryRentVehicle(int customerId, int vehicleId, DateTime dateFrom, DateTime dateTo)** Versucht, ein bestimmtes Auto für einen bestimmten Kunden zu reservieren, also einen Eintrag in der Tabelle *Vermietungen* zu erstellen. Liefert true, wenn der Eintrag angelegt werden konnte. Liefert false, wenn das Auto in diesem Zeitraum schon reserviert wurde.

Testcases:

- `TryRentVehicle(1, 17, 2020-09-01 13:00:00, 2020-09-03 13:00:00)` liefert true
- `TryRentVehicle(1, 17, 2021-01-23 13:00:00, 2021-01-25 13:00:00)` liefert false
- `TryRentVehicle(1, 17, 2021-01-28 13:00:00, 2021-01-31 13:00:00)` liefert false
- `TryRentVehicle(1, 17, 2021-01-30 13:00:00, 2021-01-31 13:00:00)` liefert true

**CalculateMonthlyIncome()** liefert eine DTO Klasse mit folgenden Informationen zurück:

 Umsatzstatistik
Jahr : Integer
Monat : Integer
Umsatz : Decimal

## Teilthema Presentationlayer / REST API

Implementieren Sie einen RentalController, der folgende Methoden unterstützt:

### GET /api/rental/cars/[Kennzeichen]

Liefert ein JSON Dokument mit allen Reservierungen des angegebenen Autos oder HTTP Status 404, wenn es nicht gefunden wurde.

#### Response

```
{
  Kennzeichen: 'W96553HU',
  Typ: 'Pkw',
  VerliehenAb: '2020-05-19 14:49:17',
  VerliehenBis: '2020-05-25 14:49:17'
}
```

### POST /api/rental/cars/[Kennzeichen]

Trägt eine Reservierung in die Datenbank ein. Dafür kann die im vorigen Punkt geschriebene Servicemethode verwendet werden. Falls diese nicht funktioniert, kann auch direkt ohne Überprüfung in die Datenbank geschrieben werden. Liefert HTTP Status 404, wenn das Autokennzeichen gefunden wurde.

#### Request Body (*application/json*)

```
{
  KundeId: 1,
  VerliehenAb: '2021-05-19 13:00:00',
  VerliehenBis: '2021-05-25 13:00:00'
}
```

### PUT /api/rental/cars/[Kennzeichen]

Trägt das aktuelle Datum in die Spalte *VerliehenBis* sowie den übermittelten Rechnungsbetrag ein, wenn der Kunde das Fahrzeug tatsächlich zurückbringt. Liefert HTTP Status 404, wenn das Autokennzeichen gefunden wurde.

#### Request Body (*application/json*)

```
{  
  KundeId: 1,  
  VerliehenAb: '2021-05-19 13:00:00',  
  Rechnungsbetrag: 214.10  
}
```