Car Shop Database – Dokumentation

Skapa en databas med bilar, lånebilar, kunder och kundbilar.

Introduktion

Denna dokumentation beskriver en simpel databas med tema bilverkstad. Den innehåller alla tabeller, relationer, SQL-kommandon, LINQ-motsvarigheter, säkerhetsaspekter och reflektioner.

ER-Diagram finns som separat fil i repo.

Customer

- **Id (PrimaryKey)**: Varje kund får ett unikt id, detta id sparas och används om de kommer in flera gånger. Används också för att koppla kunden till sin bil och en lånebil.
- **Name (TEXT NOT NULL)**: Kundens namn.
- **PhoneNumber (TEXT NOT NULL)**: Kundens telefonnummer för att kunna kontakta dem.
- **Payment (BOOL)**: Om betalningen är gjord eller inte.

På

---

phonenumber använder vi TEXT NOT NULL och inte INT eftersom ett telefonnummer kan innehålla både siffror och andra tecken som + osv.

TEXT NOT NULL eftersom dessa fält måste fyllas i.

CustomerCar

- **Id (PrimaryKey)**: Varje kundbil får ett unikt id som är baserat på bilen, dvs. bilen har samma id oavsett hur många gånger den kommer till verkstaden.
- **CarId (ForeignKey)**: Kopplar bilen till tabellen med teknisk data (årsmodell, modell, etc.).
- **CustomerId (ForeignKey)**: Vilken kund bilen tillhör.
- **Task (TEXT NOT NULL)**: Beskrivning av vad som måste göras på bilen.
- **Price (Double)**: Hur mycket reparationen/arbetet kommer kosta.
- **ETA (DATETIME)**: Ett ungefärligt datum när bilen är färdig.

CustomerCar skiljs från Car eftersom vi använder Car för alla bilar både kundernas och företagets.
TEXT NOT NULL eftersom dessa fält måste fyllas i.

CustomerRental

- **Id (PrimaryKey)**: Varje lånebil får ett unikt id.
- **CarId (ForeignKey)**: Kopplar lånebilen med tabellen för teknisk data om bilen.
- **StartDate (DATETIME)**: När bilen lämnades in.
- **EndDate (DATETIME)**: När bilen lämnades tillbaka.
- **CustomerId (ForeignKey)**: Kundens Id för att veta vem och när som använt en lånebil.

Car

- **Id (PrimaryKey)**: Varje bil får ett unikt Id.
- **Model (TEXT NOT NULL)**: Vad det är för bilmodell.
- **Brand (TEXT NOT NULL)**: Bilens märke.
- **Year (INT)**: Bilens årmodell.
- **RegNumber (TEXT NOT NULL, UNIQUE)**: Bilen registreringsnummer.

TEXT NOT NULL eftersom dessa fält måste fyllas i.

UNIQUE för att varje registreringsnummer är unikt.

Relationer

- **Customer – 1:N – CustomerCar**: En bil eller flera olika bilar kan lämnas in flera gånger av en exakt kund genom CustomerId.
- **Customer – 1:N – CustomerRental**: En kund kan få flera hyrbilar vid olika tillfällen, bilarna kopplas till en kund genom deras CustomerId.
- **Car – 1:N – CustomerCar**: Samma bil kan lämnas in flera gånger vid olika tillfällen. Varje tillfälle kopplas till en bil exakt.
- **Car – 1:N – CustomerRental**: En bil kan lånas ut flera gånger vid olika tillfällen.

SQL-Commands

```
CREATE DATABASE CarShopDB;
GO
USE CarShopDB;
GO

CREATE TABLE Customer (
Id INT PRIMARY KEY,
Name NVARCHAR(100) NOT NULL,
PhoneNumber TEXT NOT NULL,
Payment BIT NOT NULL DEFAULT 0
);
```

```sql
CREATE TABLE Car (
Id INT PRIMARY KEY,
Model NVARCHAR(100) NOT NULL,
Brand NVARCHAR(100) NOT NULL,
Year INT,
RegNumber NVARCHAR(20) NOT NULL UNIQUE
);

CREATE TABLE CustomerCar (
Id INT PRIMARY KEY,
CarId INT NOT NULL,
CustomerId INT NOT NULL,
Task NVARCHAR(100) NOT NULL,
Price DECIMAL (10, 2),
ETA DATETIME,
FOREIGN KEY (CarId) REFERENCES Car(Id),
FOREIGN KEY (CustomerId) REFERENCES Customer(Id)
);
```

DECIMAL(10,2) max 10 tecken, 2 decimal

```sql
CREATE TABLE CustomerRental (
Id INT PRIMARY KEY,
CarId INT NOT NULL,
CustomerId INT NOT NULL,
StartDate DATETIME,
EndDate DATETIME,
FOREIGN KEY (CarId) REFERENCES Car(Id),
FOREIGN KEY (CustomerId) REFERENCES Customer(Id)
);

DELETE FROM Customer WHERE Id IN (3, 4, 5, 9, 10);

INSERT INTO Car (Id, Model, Brand, Year, RegNumber)
VALUES
(1, 'E90 325i', 'BMW', 2009, 'JNM318'),
(2, 'A4 2.0 TDI', 'Audi', 2015, 'ABC123'),
(3, 'V70 D4', 'Volvo', 2013, 'XYZ456'),
(4, 'Golf GTI', 'Volkswagen', 2018, 'KLM789'),
(5, 'Civic Type R', 'Honda', 2020, 'HND321'),
(6, 'Model 3', 'Tesla', 2022, 'ELC555'),
(7, 'Corsa 1.4', 'Opel', 2011, 'OPL777'),
(8, 'Fiesta 1.0 EcoBoost', 'Ford', 2017, 'FRD123'),
(9, 'Impreza WRX', 'Subaru', 2014, 'SUB555'),
(10, '500 1.2', 'Fiat', 2012, 'FAT900');

INSERT INTO Customer (Id, Name, PhoneNumber, Payment)
VALUES
```

```
(1, 'Olof Svensson', '0701234567', 0),
(2, 'Anna Karlsson', '0708112233', 0),
(3, 'Erik Johansson', '0739998877', 0),
(4, 'Sara Lindgren', '0722554433', 0),
(5, 'Johan Persson', '0765443322', 0),
(6, 'Lisa Berg', '0704567890', 0),
(7, 'Marcus Nilsson', '0719876543', 0),
(8, 'Emilia Andersson', '0732345678', 0);

INSERT INTO CustomerCar (Id, CarId, CustomerId, Task, Price, ETA)
VALUES
(1, 2, 2, 'Service', 1295.00, '2025-12-09 10:00'),
(2, 3, 6, 'Inspection', 899.00, '2025-12-09 09:00'),
(3, 4, 7, 'Timingchain replacement', 8990.00, '2025-12-13 16:00'),
(4, 5, 8, 'Tyre replacement', 3495.00, '2025-12-09 11:00'),
(5, 1, 1, 'Suspension replacement', 5000.00, '2025-12-11 13:00');

INSERT INTO CustomerRental (Id, CarId, CustomerId, StartDate, EndDate)
VALUES
(1, 6, 1, '2025-12-09 09:00', '2025-12-11 13:00'),
(2, 7, 2, '2025-12-09 09:00', '2025-12-09 10:00'),
(3, 8, 6, '2025-12-09 07:00', '2025-12-09 09:00'),
(4, 9, 7, '2025-12-11 07:00', '2025-12-13 16:00'),
(5, 10, 8, '2025-12-09 07:00', '2025-12-09 11:00');
```

SQL Frågor Exempel

- Vilka reparationer kostar över 2000kr?

```
SELECT Task, Price FROM CustomerCar WHERE Price > 2000;
```

- Vilka bilar i verkstaden är äldre än 2013?

```
SELECT * FROM Car WHERE Year < 2013;
```

- Sortera efter registreringsnummer

```
SELECT * FROM Car ORDER BY RegNumber;
```

- Hitta alla BMW-bilar

```
SELECT * FROM Car WHERE Brand LIKE 'BMW';
```

- Hur mycket varje kund spenderat

```
SELECT CustomerId, SUM(Price) AS TotalRepairCost FROM CustomerCar GROUP BY
CustomerId;
```

- Vilken kund har vilken lånebil

```
SELECT CarId, CustomerId FROM CustomerRental ORDER BY CarId;
```

- Vem har kört lånebil längst

```sql
SELECT c.Name AS Kund, car.Brand + ' ' + car.Model AS LåneBil,
DATEDIFF(HOUR, cr.StartDate, cr.EndDate) AS Timmar,
cr.StartDate AS Från, cr.EndDate AS Till
FROM CustomerRental cr
JOIN Customer c ON c.Id = cr.CustomerId
JOIN Car car ON car.Id = cr.CarId
ORDER BY DATEDIFF(HOUR, cr.StartDate, cr.EndDate) DESC;
```

- Vem äger vilken bil?

```sql
SELECT c.Name AS CustomerName, car.Brand, car.Model
FROM Customer c
JOIN CustomerCar cc ON c.Id = cc.CustomerId
JOIN Car car on cc.CarId = car.Id
ORDER BY c.Name;
```

LINQ Exempel

```csharp
// Ta bort kunder med specifika Ids
var idsToDelete = new List<int> { 3, 4, 5, 9, 10 };
var customersToDelete = context.Customers.Where(c =>
idsToDelete.Contains(c.Id));
context.Customers.RemoveRange(customersToDelete);

// Hitta dyra serviceärenden
var expensiveTasks = context.CustomerCars
    .Where(c => c.Price > 2000)
    .Select(c => new { c.Task, c.Price })
    .ToList();

// Skapa ny kund
var Olof = new Customer {
  Id = 1,
  Name = "Olof Svensson",
  PhoneNumber = "0701234567",
  Payment = false
};
context.Customers.Add(Olof);
```

Säkerhet Databaser

Databaser används för att spara mängder av information som används för att vi människor ska kunna använda teknologi och ta bättre beslut. Det betyder också att det finns personer som vill använda denna information på ett olagligt eller omoraliskt sätt för egen vinning, vilket kan skada verksamheten/personen som hanterar informationen, samt alla personer vars information ligger i databasen.

Därför behöver det finnas lager som ser till att endast de som ska ha tillgång till en databas faktiskt får det.

I en databas som den skapad i denna uppgift hade man sett till så att mekaniker och butiksanställda har tillgång att skapa och ändra befintliga objekt i databasen (Read, Update). Den enda som skulle kunna ändra och skapa nya tabeller hade varit chefen (Create, Read, Update, Delete). Receptionister hade endast kunna läsa av och ändra små saker somtiden bilen lämnades osv (Read, Update). Inga kunder får tillgång till databasen, bara genom t.ex. sms utskick när statusen på bilen är klar (ej implenterat).

Autentisering (Authentication) används för att säkerställa att den som försöker använda en tjänst är rätt person. Tvåfaktorautentisering används genom att skicka ett meddelande eller notis till en annan enhet eller e-post där användaren kan bekräfta om det är de som försöker logga in eller inte. En annan metod som används på många hemelektronikprodukter är fingeravtrycksläsare och ansiktsigenkänning. Ditt fingeravtryck och konturerna i ditt ansikte är helt unika för dig, så detta är ett sätt att autentisera att det verkligen är du som försöker komma åt något.

Auktorisering (Authorization) används för att hålla koll på vad en autentiserad användare får göra. T.ex. kan en användare på en webbsida redigera sin egen profil eller scrolla i sitt flöde, men de kan inte redigera andras profiler. Auktorisering håller alltså koll på vem som får göra vad. I en databas är det viktigt att skilja på vilka som får läsa från den och vilka som kan skriva eller redigera informationen.

Reflektion

Att skriva i SQL och få databasen att fungera är kul och väldigt straight forward, men det jag tycker är klurigt är förarbetet när man ska bestämma vad som ska innehålla vad, hur man ska relatera klasser till varandra osv. Nu var det enkelt eftersom jag kunde använda id för att koppla ihop allt, men jag antar att det finns fall där detta inte går.

En sak som kan förbättras är det visuella i det här dokumentet. Annars tror jag att jag saknar en del djupgående kunskap när det kommer till databaser, och därför blev denna inlämning ganska basal.

Databasen kan skalas genom att lägga till en ny tabell där man kan spara varje order och besök med pris, vad som gjordes och vem kunden var. På det sätt kan man sedan söka på CustomerId eller namn och se vad kunderna har gjort hos er innan. Om man kopplar databasen till ett program som tar hand om fakturor kan man ha en boolean som faktura betald true/false som bockas in så fort fakturan är betald. Finns massa annat man kan lägga till, automatiskt mejl eller sms system som skickar sms så fort man bockar av att bilen är klar. Då kan en boolean i databasen aktiverats och sedan skickas det till ett program som skickar sms osv.

När det gäller skillnaden mellan SQL och LINQ kan LINQ användas för att hantera data inom List<> Dictionary<>, men när datamängder växer och relationer blir fler kan det bli svårare att ha koll på alla information.