M153 Datenbank Projekt

AUTO LEASING

Thierry Lötscher & Fabrizio Poli

19.04.2023

1 Inhaltsverzeichnis

2 Kurzbeschreibung

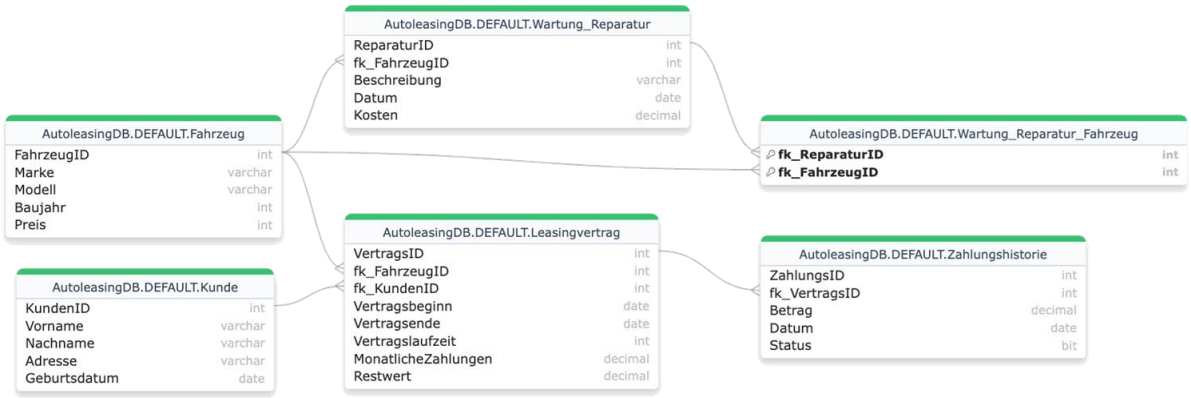
Wir haben eine Datenbank für ein mögliches Auto Leasing Unternehmen gemacht. Die Datenbank beinhaltet Kundendaten sowie Informationen über alle Zahlungen oder geleasten Autos welche über einfache Abfragen oder Funktionen / Stored Procedures geholt werden können. Wir haben uns vor allem auf Luxusautos fokussiert.

3 Abgegebene Dateien

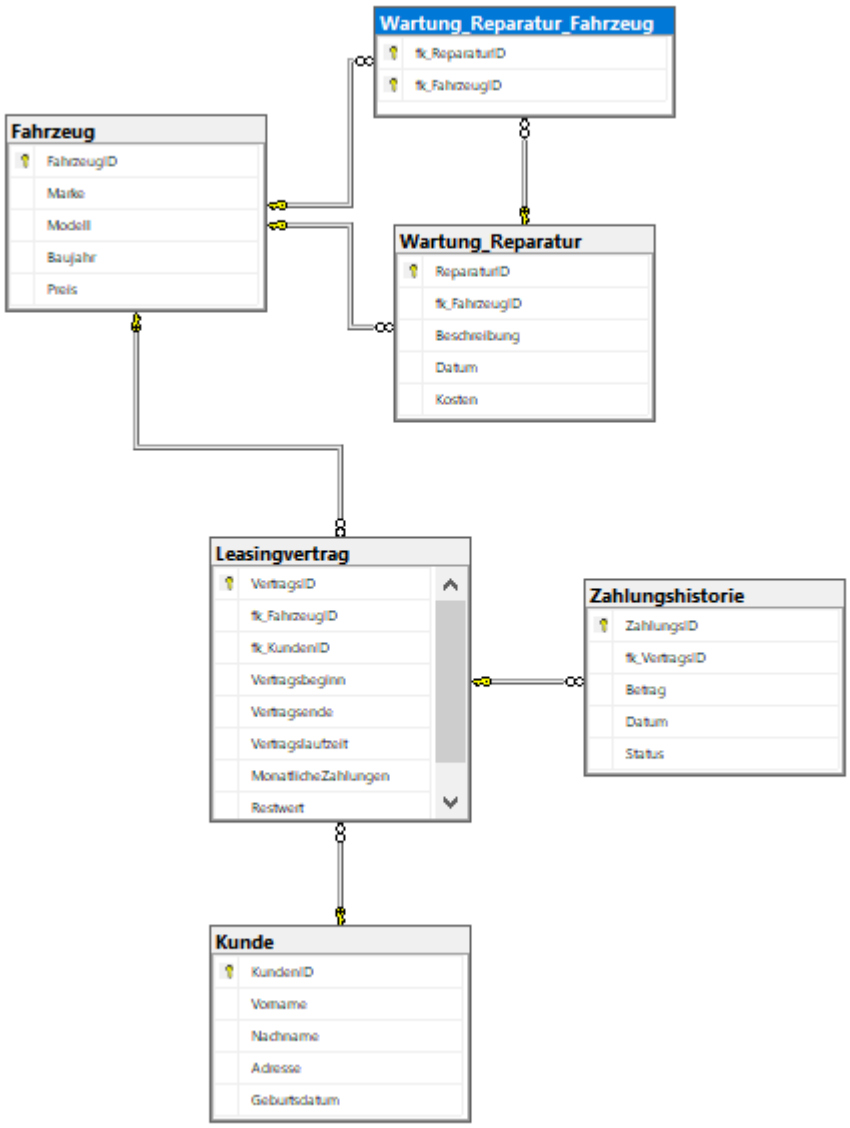
3.1 M153_AutoLeasing_Lötscher_Thierry_Poli_Fabrizio.zip:

- M153_AutoLeasing.docx
- M153_AutoLeasing_Insert.sql
- M153_AutoLeasing_Create.sql
- M153_AutoLeasing_fn_GetKundenFahrzeuge.sql
- M153_AutoLeasing_fn_GetDurchschnittsZahlungen.sql
- M153_AutoLeasing_sp_AddFahrzeug.sql
- M153_AutoLeasing_sp_GetLeasingVertragDetails.sql

4 ER-Diagramm AutoLeasing Datenbank



5 Relationales Modell AutoLeasing Datenbank



6 Funktionen Beschreibung und Anwendung

6.1 GetDurchschnittlicheZahlungenNachMarke

Mit dieser Funktion kann man die durchschnittliche Zahlung nach Automarke herauslesen. Es wird eine Zwischentabelle erstellt in der die SELECT Abfrage gespeichert wird.

- File: M153_AutoLeasing_fn_GetDurchschnittlicheZahlungenNachMarke.sql

```
-- ************************************************************************
-- * Function GetDurchschnittlicheZahlungenNachMarke erstellen.
-- ************************************************************************
CREATE FUNCTION GetDurchschnittlicheZahlungenNachMarke ()
RETURNS @Table TABLE ([Marke] varchar(80), [DurchschnittlicheZahlung] decimal(9,2))
AS
BEGIN
    -- Überprüfung, ob Daten vorhanden sind
    IF (NOT EXISTS (SELECT 1 FROM Fahrzeug) OR NOT EXISTS (SELECT 1 FROM Leasingvertrag))
    BEGIN
        -- Keine Daten vorhanden, NULL als Ergebnis zurückgeben
        RETURN;
    END

    -- Die Funktion gibt eine Tabelle mit zwei Spalten zurück: "Marke" und "DurchschnittlicheZahlung"
    -- Eine temporäre Tabelle @Table wird erstellt, um die Ergebnisse zu speichern
    -- Die Spalten "Marke" und "DurchschnittlicheZahlung" werden in der Tabelle definiert.
    INSERT INTO @Table (Marke, DurchschnittlicheZahlung)
    SELECT F.Marke, AVG(LV.MonatlicheZahlungen) AS DurchschnittlicheZahlungen
    FROM Fahrzeug F
    INNER JOIN Leasingvertrag LV ON F.FahrzeugID = LV.fk_FahrzeugID
    GROUP BY F.Marke;
    RETURN;
END;
go
```

6.1.1 GetDurchschnittlicheZahlungenNachMarke Anwendung

```
SELECT * FROM GetDurchschnittlicheZahlungenNachMarke();
```

6.2 GetKundeFahrzeuge

Wenn man diese Funktion auführt wird eine Tabelle zurück gegeben bei der man die Kunden mit den geleasten Autos sieht.

- File: M153_AutoLeasing_fn_ GetKundeFahrzeuge.sql

```sql
-- ************************************************************************
-- * Function GetKundeFahrzeuge erstellen.
-- ************************************************************************
CREATE FUNCTION GetKundeFahrzeuge (
    @KundenID INT
)
RETURNS TABLE
AS
RETURN
(
    -- Die Funktion gibt eine Tabelle zurück, daher wird der SELECT-Anweisung eine Tabelle zugewiesen.
    BEGIN
        SELECT K.KundenID, K.Vorname, K.Nachname, K.Adresse, K.Geburtsdatum,
            COUNT(LV.VertragsID) AS AnzahlFahrzeuge,
            STUFF(
                (
                    -- Die STUFF-Funktion wird verwendet, um die Liste der gemieteten Fahrzeuge zu generieren.
                    -- Die FOR XML PATH-Klausel dient zum Konkatenieren der Fahrzeugmarken und -modelle.
                    SELECT ', ' + F.Marke + ' ' + F.Modell
                    FROM Leasingvertrag LV
                    INNER JOIN Fahrzeug F ON LV.FahrzeugID = F.FahrzeugID
                    WHERE LV.FahrzeugID = K.KundenID
                    FOR XML PATH(''), TYPE
                ).value('.', 'NVARCHAR(MAX)'), 1, 2, ''
            ) AS GemieteteFahrzeuge
        FROM Kunde K
        LEFT JOIN Leasingvertrag LV ON K.KundenID = LV.FahrzeugID
        WHERE K.KundenID = @KundenID
        GROUP BY K.KundenID, K.Vorname, K.Nachname, K.Adresse, K.Geburtsdatum
    END
);
go
```

6.2.1 GetKundeFahrzeuge Anwendung

```sql
SELECT * FROM GetKundeFahrzeuge(1);
```

7 Stored Procedures Beschreibung und Anwendung

7.1 AddFahrzeug

Mit diesem Stored Procedure kann man ein Auto zur Fahrzeug Tabelle hinzufügen welches dann geleased werden kann. Man muss die Marke, Modell, Baujahr und den Preis als Parameter mitgeben. Es wird auch überprüft, ob die Parameter nicht NULL sind.

- File: M153_AutoLeasing_sp_AddFahrzeug.sql

```sql
-- ************************************************************************
-- * Stored Procedure AddFahrzeug erstellen.
-- ************************************************************************
CREATE PROCEDURE AddFahrzeug
    @Marke VARCHAR(50),
    @Modell VARCHAR(50),
    @Baujahr INT,
    @Preis INT
AS
BEGIN
    -- Überprüfen, ob alle Parameter befüllt sind
    IF @Marke IS NULL
    BEGIN
        RAISERROR('Der Parameter @Marke darf nicht NULL sein.', 16, 1)
        RETURN
    END

    IF @Modell IS NULL
    BEGIN
        RAISERROR('Der Parameter @Modell darf nicht NULL sein.', 16, 1)
        RETURN
    END

    IF @Baujahr IS NULL
    BEGIN
        RAISERROR('Der Parameter @Baujahr darf nicht NULL sein.', 16, 1)
        RETURN
    END

    IF @Preis IS NULL
    BEGIN
        RAISERROR('Der Parameter @Preis darf nicht NULL sein.', 16, 1)
        RETURN
    END

    -- Insert-Anweisung ausführen
    INSERT INTO Fahrzeug (Marke, Modell, Baujahr, Preis)
    VALUES (@Marke, @Modell, @Baujahr, @Preis)
    RETURN 1
END
go
```

7.1.1 AddFahrzeug Anwendung

```sql
EXEC AddFahrzeug @Marke = 'Ford', @Modell = 'Mustang', @Baujahr = 1965, @Preis = 77423;
```

7.2 GetLeasingVertragDetails

Mit diesem Stored Procedure bekommt man alle Details zu einem Vertrag. Als Parameter muss die VertragsId mitgegeben werden. Falls der Parameter NULL ist erscheint eine Error Message die darauf hinweist.

- File: M153_AutoLeasing_sp_GetLeasingVertragDetails.sql

```
-- ******************************************************************
-- * Stored Procedure GetLeasingVertragDetails erstellen.
-- ******************************************************************
CREATE PROCEDURE GetLeasingVertragDetails
    @VertragsID INT
AS
BEGIN
    -- Überprüfen, ob der Parameter befüllt ist
    IF @VertragsID IS NULL
    BEGIN
        RAISERROR('Der Parameter @VertragsID darf nicht NULL sein.', 16, 1)
        RETURN
    END

    -- Abfrage ausführen
    SELECT LV.VertragsID, F.Marke, F.Modell, K.Vorname, K.Nachname, LV.Vertragsbeginn, LV.Vertragsende,
           LV.Vertragslaufzeit, LV.MonatlicheZahlungen, LV.Restwert,
           Z.ZahlungsID, Z.Betrag, Z.Datum, Z.Status
    FROM Leasingvertrag LV
    INNER JOIN Fahrzeug F ON LV.fk_FahrzeugID = F.FahrzeugID
    INNER JOIN Kunde K ON LV.fk_KundenID = K.KundenID
    LEFT JOIN Zahlungshistorie Z ON LV.VertragsID = Z.fk_VertragsID
    WHERE LV.VertragsID = @VertragsID;
END
go
```

7.2.1 GetLeasingVertragDetails Anwendung

```
EXEC GetLeasingVertragDetails
  @VertragsID = 11;
```

8 Beschreibung der Tests

8.1 GetDurchschnittlicheZahlungenNachMarke

In diesen Test wird die Funktion GetDurchschnittlicheZahlungenNachMarke einmal normal ausgeführt und einmal mit spezifischen Selektoren.

```
-- Test 1.0: Führt die Function GetDurchschnittlicheZahlungenNachMarke
-- aus.
SELECT * FROM GetDurchschnittlicheZahlungenNachMarke();

-- Test 2.0: Führt die Function GetDurchschnittlicheZahlungenNachMarke
-- aus mit spezifischen spalten.
SELECT Marke, DurchschnittlicheZahlung FROM GetDurchschnittlicheZahlungenNachMarke();
```

8.2 GetKundeFahrzeuge

In den drei Tests wird die Funktion GetKundeFahrzeuge ausgeführt mit verschiedenen Parameter.

```sql
-- Test 1.0: Führt die Funktion aus und gibt die Fahrezeuge aus
-- welche der Kunde mit der Id 1 gemietet hat.
SELECT * FROM GetKundeFahrzeuge(1);

-- Test 2.0: Führt die Funktion aus und gibt die Fahrezeuge aus
-- welche der Kunde mit der Id 3 gemietet hat.
SELECT * FROM GetKundeFahrzeuge(3);

-- Test 3.0: Führt die Funktion aus und gibt die Fahrezeuge aus
-- welche der Kunde mit der Id 5 gemietet hat.
SELECT * FROM GetKundeFahrzeuge(5);
```

8.3 AddFahrzeug

In den ersten zwei Tests wird das Stored Procedure AddFahrzeug mit verschiedenen Parameter ausgeführt und zum Schluss noch getestet ob die Daten in der Datenbank sind. Der dritte Test soll Error Meldungen auslösen.

```sql
-- Test 1.0: Fügt den Ford Mustang zur Fahrzeug Tabelle hinzu und
-- Testet ob der Ford Mustang vorhanden ist.
EXEC AddFahrzeug @Marke = 'Ford', @Modell = 'Mustang', @Baujahr = 1965, @Preis = 77423;
SELECT * FROM Fahrzeug WHERE Marke = 'Ford' AND Modell = 'Mustang';

-- Test 2.0: Fügt den Dodge Charger zur Fahrzeug Tabelle hinzu und
-- Testet ob der Dodge Charger vorhanden ist.
EXEC AddFahrzeug @Marke = 'Dodge', @Modell = 'Charger', @Baujahr = 1971, @Preis = 89900;
SELECT * FROM Fahrzeug WHERE Marke = 'Dodge' AND Modell = 'Charger';

-- Test 3.0: Gibt ein Error zurück das die Parameter nicht NULL sein dürfen.
EXEC AddFahrzeug @Marke = NULL, @Modell = NULL, @Baujahr = NULL, @Preis = NULL;
```

8.4 GetLeasingVertragDetails

In den ersten zwei Tests wird das Stored Procedure GetLeasingVertragDetails mit verschiedenen Parameter ausgeführt. Der dritte Test soll Error Meldungen auslösen.

```sql
-- Test 1.0: Führt das Stored Procedure aus und gibt die
-- LeasingVertragDetails aus des Vetrags mit der Id 11.
EXEC GetLeasingVertragDetails
  @VertragsID = 11;

-- Test 2.0: Führt das Stored Procedure aus und gibt die
-- LeasingVertragDetails aus des Vetrags mit der Id 13.
EXEC GetLeasingVertragDetails
 @VertragsID = 13;

-- Test 3.0: Führt das Stored Procedure aus und gibt ein Error zurück
-- weil der Parameter NULL ist.
EXEC GetLeasingVertragDetails
  @VertragsID = NULL;
```