Car Dealership

WINTER SEMESTER 2023/2024

LUKASZ WUTKOWSKI

Database Project Description `Car_dealership`

`Car_dealership` is a database designed to manage operations within a car dealership. The system enables effective tracking and management of key aspects of the dealership's activities, including vehicle inventory, sales transactions, customer information, and service actions. The aim of this project is to streamline business processes, ensure data accuracy, and facilitate business decision-making by providing quick access to essential information.

Key Functionalities:

1. **Vehicle Inventory Management:** The database allows for accurate tracking of all vehicles available at the dealership, along with their detailed specifications, such as make, model, year of manufacture, engine type and power, as well as condition (new/used) and mileage.

2. **Sales Management**: The system records all sales transactions, enabling tracking of vehicles sold, sales prices, and assigning sales to specific dealership employees, which allows for analysis of sales performance.

3. **Customer Information:** The database stores detailed information about customers, including their contact and address details, facilitating customer relationship building and management of marketing activities.

4. **Service and Maintenance:** The system allows for the recording and management of service actions performed on vehicles, including service dates, descriptions of work done, and costs.

5. **Audit and Monitoring:** With an `AuditLog` table, the project provides the capability to track significant changes in the database, thereby increasing security and user accountability.

Creating the Database. SQL Statements:

In this section, SQL statements are provided for the purpose of creating and defining the structure of the `Car_dealership` database. The SQL statements are used to create a new database and to design and implement tables that will store all the key information related to the operations of the car dealership. Each table is intended to hold data specific to different aspects of the business, such as vehicle data, employee information, sales transactions, customer data, inventory stocks, vehicle services, and audit logs.

```
CREATE DATABASE Car_dealership

-- The Cars table stores information about the vehicles available at the
dealership.
-- Each car is uniquely identified by its VIN (Vehicle Identification Number).
CREATE TABLE Cars (
    VIN VARCHAR(50) PRIMARY KEY, -- Unique Vehicle Identification Number
    Brand VARCHAR(50) NOT NULL,
    Model VARCHAR(50) NOT NULL,
    YearOfManufacture INT NOT NULL,
    EngineType VARCHAR(50) NOT NULL, --Engine type (e.g. Diesel, Gasoline, Hybrid,
Electric)
    EngineCapacity DECIMAL(5, 2) NOT NULL, -- Engine Capacity. For electric
vehicles, the battery capacity should be specified, expressed in kWh.
    EnginePower INT NOT NULL, -- Engine Power expressed in HP (Horsepower)
    IsNew BIT NOT NULL, -- 1 indicates that the car is new, 0 indicates that the
car is used.
    Mileage INT -- Car mileage in kilometers
);

-- The `Employees` table contains information about the dealership's employees.
-- Each employee is identified by a unique EmployeeID.
CREATE TABLE Employees (
    EmployeeID INT PRIMARY KEY IDENTITY(1,1), -- Unique Employee Identifier
    FirstName VARCHAR(50) NOT NULL,
    LastName VARCHAR(50) NOT NULL,
    Position VARCHAR(50) NOT NULL,
    Email VARCHAR(100) NOT NULL,
    PhoneNumber VARCHAR(15) NOT NULL,
    StreetAddress VARCHAR(255) NOT NULL, -- Detailed street address along with
apartment or house number
    City VARCHAR(100) NOT NULL,
    PostalCode VARCHAR(50) NOT NULL,
    Country VARCHAR(50) NOT NULL,
    BaseSalary MONEY NOT NULL, -- Employee's monthly salary
    HireDate DATE NOT NULL -- Employee's employment date
);

-- The `Customers` table stores information about the dealership's clients.
-- Each customer is identified by a unique CustomerID.
CREATE TABLE Customers (
    CustomerID INT PRIMARY KEY IDENTITY(1,1), -- Unique Customer Identifier
```

```
    FirstName VARCHAR(50) NOT NULL,
    LastName VARCHAR(50) NOT NULL,
    Email VARCHAR(100),
    PhoneNumber VARCHAR(50), -- Customer's phone number, including the country code
for international customers.
    StreetAddress VARCHAR(255) NOT NULL, -- Detailed street address along with the
apartment or house number.
    City VARCHAR(255) NOT NULL,
    PostalCode VARCHAR(50) NOT NULL,
    Country VARCHAR(50) NOT NULL,
);

-- The `Inventories` table monitors the status of cars in the dealership's stock.
-- Each entry is identified by a unique InventoryID.
CREATE TABLE Inventories (
    InventoryID INT PRIMARY KEY IDENTITY(1,1), -- Unique inventory entry
identifier.
    VIN VARCHAR(50) NOT NULL, -- Foreign key to the `Cars` table, cannot be null.
    Status VARCHAR(50) NOT NULL, -- Car status ('Available', 'Sold' or 'In
    Repair').
    AcquisitionDate DATE NOT NULL, -- Date the dealership acquired the car.
    AcquisitionPrice MONEY NOT NULL, -- Purchase price of the car by the
dealership.
    Price MONEY NOT NULL, -- Car price.
    CONSTRAINT FK_Inventories_Cars FOREIGN KEY (VIN) REFERENCES Cars(VIN)
);

-- The `Sales` table records information about car sales.
-- Each sale is identified by a unique SaleID.
CREATE TABLE Sales (
    SaleID INT PRIMARY KEY IDENTITY(1,1), -- Unique sale identifier.
    CustomerID INT NOT NULL, -- Foreign key to the `Customers` table, cannot be
null.
    VIN VARCHAR(50) NOT NULL, -- Foreign key to the `Cars` table, cannot be null.
    EmployeeID INT NOT NULL, -- Foreign key to the `Employees` table, cannot be
    null.
    SaleDate DATE NOT NULL, -- Sale date, cannot be null.
    SalePrice MONEY NOT NULL -- Sale price, cannot be null.
    CONSTRAINT FK_Sales_Customers FOREIGN KEY (CustomerID) REFERENCES
Customers(CustomerID),
    CONSTRAINT FK_Sales_Cars FOREIGN KEY (VIN) REFERENCES Cars(VIN),
    CONSTRAINT FK_Sales_Employees FOREIGN KEY (EmployeeID) REFERENCES
Employees(EmployeeID)
);

-- The `Services` table records information about services performed on cars.
-- Each service is identified by a unique ServiceID.
CREATE TABLE Services (
    ServiceID INT PRIMARY KEY IDENTITY(1,1), -- Unique service identifier.
    VIN VARCHAR(50) NOT NULL, -- Foreign key to the `Cars` table, cannot be null.
    ServiceDate DATE NOT NULL,
```

```
    Description VARCHAR(255) NOT NULL, -- Service description, containing
information about the type of repair.
    Cost MONEY NOT NULL -- Service cost, cannot be null.
    CONSTRAINT FK_Services_Cars FOREIGN KEY (VIN) REFERENCES Cars(VIN)
);

-- The `AuditLog` table is used to track important operations performed on the
database.
CREATE TABLE AuditLog (
    AuditID INT PRIMARY KEY IDENTITY(1,1), -- Unique identifier of the performed
change.
    TableName VARCHAR(100) NOT NULL, -- Name of the table where the change was
made.
    OperationType VARCHAR(50) NOT NULL, -- Type of operation (e.g., INSERT, UPDATE,
DELETE).
    OperationDetails NVARCHAR(MAX), -- Detailed description of the operation.
    OperationDate DATETIME NOT NULL DEFAULT GETDATE(), -- Date and time of the
operation.
    UserName NVARCHAR(128), -- Username of the user performing the operation.
);
```

CHECK constraints:

```
-- Constraint for the 'Inventories' table - Acquisition Date (AcquisitionDate).
-- The purpose of the constraint is to ensure that the acquisition date is not
later than the current system date.
ALTER TABLE dbo.Inventories
ADD CONSTRAINT CHK_Inventories_AcquisitionDate CHECK (AcquisitionDate <=
GETDATE());

-- Constraint for the 'Customers' table - email format.
-- The purpose of the constraint is basic validation, which involves checking
whether the email address contains the '@' symbol.
ALTER TABLE dbo.Customers
ADD CONSTRAINT CHK_Customers_Email CHECK (Email LIKE '%@%');


-- Constraint for the 'Inventories' table – Status.
-- The purpose of the constraint is to ensure consistency and precision in
determining the status of the vehicle.
-- The car status can take one of three predefined values: 'Available', 'Sold', 'In
Repair'.
ALTER TABLE Inventories
ADD CONSTRAINT CHK_Inventories_Status CHECK (Status IN ('Available', 'Sold', 'In
Repair'));
```

VIEWS:

```
-- The view 'AvailableCars' provides quick access to information about all
available cars in the dealership.
```

```sql
-- It combines data from the 'Cars' and 'Inventories' tables to display key
information about each car whose status is marked as 'Available'.
-- Usage of the view: SELECT * FROM AvailableCars;
CREATE VIEW AvailableCars AS
SELECT Cars.VIN, Cars.Brand, Cars.Model, Cars.YearOfManufacture,
Inventories.Status, Inventories.Price
FROM dbo.Cars
JOIN dbo.Inventories ON Cars.VIN = Inventories.VIN
WHERE Inventories.Status = 'Available';

-- The 'SalesProfitByEmployee' view is intended for analyzing the sales performance
of employees.
-- Its purpose is to calculate the total sales profit generated by each employee.
-- A simple algorithm is based on the difference between the selling price and the
purchase price of the vehicle.
-- Usage of the view: SELECT * FROM SalesProfitByEmployee;
CREATE OR ALTER VIEW SalesProfitByEmployee AS
SELECT
    e.EmployeeID,
    e.FirstName,
    e.LastName,
    SUM(s.SalePrice - i.AcquisitionPrice) AS TotalProfit
FROM dbo.Sales s
JOIN dbo.Employees e ON s.EmployeeID = e.EmployeeID
JOIN dbo.Inventories i ON s.VIN = i.VIN
WHERE i.Status = 'Sold' -- An additional condition is added to consider only sold
vehicles.
GROUP BY e.EmployeeID, e.FirstName, e.LastName;
```

Triggers:

```sql
-- The trigger 'trg_AuditLog_Inventories' has been defined for the 'Inventories'
table.
-- Its purpose is to record significant operations (insertion, update, and
deletion) performed on this table.
-- Change records are saved in the AuditLog table.
-- by registering the table name, operation type, and detailed operation
information.
-- and the username (obtained using the SUSER_NAME() function) who made the change.
CREATE OR ALTER TRIGGER trg_AuditLog_Inventories
ON dbo.Inventories
AFTER INSERT, UPDATE, DELETE
AS
BEGIN
    DECLARE @operationType VARCHAR(50)
    DECLARE @details NVARCHAR(50) = ''
    DECLARE @recordID INT

    SET @operationType = CASE
```

```
                            WHEN EXISTS(SELECT * FROM inserted) AND EXISTS(SELECT *
FROM deleted) THEN 'UPDATE'
                            WHEN EXISTS(SELECT * FROM inserted) THEN 'INSERT'
                            WHEN EXISTS(SELECT * FROM deleted) THEN 'DELETE'
                        END;

    -- Collecting information about the type of operation on a specific record
    -- for UPDATE
    IF @operationType = 'UPDATE'
    BEGIN
        SELECT @recordID = i.InventoryID
        FROM inserted i
        SET @details = 'Updated record with InventoryID = ' + CAST(@recordID AS
NVARCHAR(50))
    END;

    -- for INSERT
    ELSE IF @operationType = 'INSERT'
    BEGIN
        SELECT @recordID = i.InventoryID
        FROM inserted i
        SET @details = 'Inserted new record with InventoryID = ' + CAST(@recordID
AS NVARCHAR(50))
    END;

    -- for DELETE
    ELSE IF @operationType = 'DELETE'
    BEGIN
        SELECT @recordID = d.InventoryID
        FROM deleted d
        SET @details = 'Deleted record with InventoryID = ' + CAST(@recordID AS
NVARCHAR(50))
    END;

    INSERT INTO dbo.AuditLog (TableName, OperationType, OperationDetails, UserName)
    VALUES ('Inventories', @operationType, @details, SUSER_NAME())
END;
```

Functions:

```
-- The 'GetCarsByBrand' function returns a list of cars of a specific brand.
-- It accepts the brand name as a parameter.
-- Usage: SELECT * FROM dbo.GetCarsByBrand('Toyota');
CREATE FUNCTION GetCarsByBrand (@BrandName VARCHAR(50))
RETURNS TABLE
AS
RETURN
(
    SELECT VIN, Brand, Model, YearOfManufacture, EngineType, EngineCapacity,
EnginePower, IsNew, Mileage
    FROM dbo.Cars
```

```
    WHERE Brand = @BrandName
);
```

Procedures:

```
-- The task of the procedure is to update the price of a car in the Inventories
table.
-- The procedure first checks whether the car with the given VIN exists in the
Inventories table before proceeding to update the price.
-- After changing the price, the procedure records the operation in the AuditLog
table (thanks to the previously created trigger, which is located in the 'Triggers'
section).
CREATE PROCEDURE dbo.UpdateCarPrice
    @VIN VARCHAR(50),
    @NewPrice MONEY
AS
BEGIN
    -- Checking if the car exists.
    IF EXISTS(SELECT 1 FROM dbo.Inventories WHERE VIN = @VIN)
    BEGIN
        -- Updating the car's price.
        UPDATE dbo.Inventories
        SET Price = @NewPrice
        WHERE VIN = @VIN;
    END
    ELSE
    BEGIN
        -- If the car does not exist, return a message.
        PRINT 'Car doesn´t exist.';
    END
END;
```

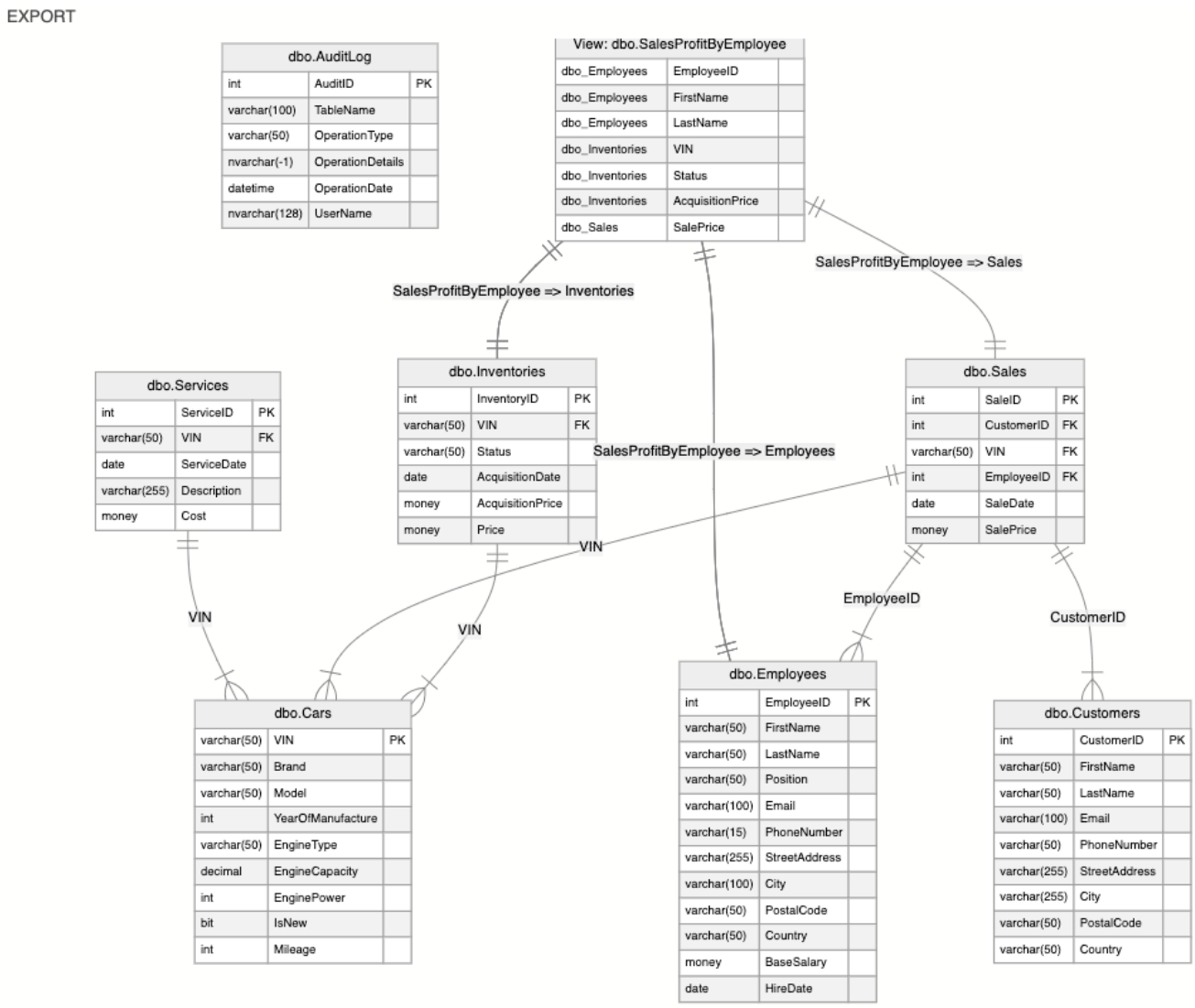
Sample data:

```
-- Table 'Cars'
INSERT INTO Cars (VIN, Brand, Model, YearOfManufacture, EngineType, EngineCapacity,
EnginePower, IsNew, Mileage)
VALUES
('VIN12345', 'Toyota', 'Corolla', 2020, 'Gasoline', 1.8, 140, 0, 50000),
('VIN12346', 'Tesla', 'Model S', 2021, 'Electric', 75, 416, 0, 120000),
('VIN12347', 'Ford', 'Focus', 2019, 'Diesel', 2.0, 150, 0, 130000),
('VIN12348', 'Volkswagen', 'Golf', 2018, 'Diesel', 2.0, 110, 0, 40000),
('VIN12349', 'Audi', 'A4', 2019, 'Gasoline', 1.4, 150, 0, 110000),
('VIN12350', 'BMW', '3', 2024, 'Hybrid', 2.5, 184, 1, 0),
('VIN12351', 'Mercedes', 'C Klasse', 2017, 'Diesel', 2.2, 170, 0, 260000),
('VIN12352', 'Tesla', 'Model Y', 2024, 'Electric', 75, 440, 1, 0),
('VIN12353', 'Toyota', 'Prius Plus', 2013, 'Hybrid', 1.8, 140, 0, 165000),
('VIN12354', 'Skoda', 'Octavia', 2021, 'Diesel', 2.0, 204, 0, 34000),
('VIN12355', 'Toyota', 'Camry', 2024, 'Hybrid', 2.0, 170, 1, 0),
('VIN12356', 'Peugeot', '508', 2021, 'Diesel', 2.0, 130, 0, 25000),
```

```
('VIN12357', 'Renault', 'Zoe', 2020, 'Electric', 41, 120, 0, 50000),
('VIN12358', 'Toyota', 'Rav4', 2024, 'Hybrid', 2.5, 180, 1, 0),
('VIN12359', 'Mercedes', 'Model E', 2021, 'Gasoline', 2.5, 250, 0, 20000);

-- Table 'Employees'.
INSERT INTO Employees (FirstName, LastName, Position, Email, PhoneNumber,
StreetAddress, City, PostalCode, Country, BaseSalary, HireDate)
VALUES
('Jan', 'Kowalski', 'Salesperson', 'jkowalski@example.com', '123456789', 'Radhus
1', 'Oslo', '0250', 'Norway', 35000, '2021-01-10'),
('Lars', 'Nygaard', 'Salesperson', 'lnygaard@example.com', '966456789', 'Kjoping
4', 'Sandvika', '0290', 'Norway', 35000, '2023-04-05'),
('Anna', 'Nowak', 'Sales Manager', 'anowak@example.com', '987654321', 'Radhus 1',
'Oslo', '0250', 'Norway', 50000, '2020-06-15');

-- Table 'Customers'.
INSERT INTO Customers (FirstName, LastName, Email, PhoneNumber, StreetAddress,
City, PostalCode, Country)
VALUES
('Lars', 'Hansen', 'lars.hansen@example.com', '11222333', 'Gamlevei 3', 'Bergen',
'5003', 'Norway'),
('Maija', 'Vestby', 'm.vest@example.com', '22332433', 'Nyveien 16', 'Oslo', '2003',
'Norway'),
('Lucia', 'Hammar', 'lu.hamm@example.com', '69825343', 'Klovre 4', 'Holmestrand',
'3080', 'Norway'),
('Anders', 'Roroar', 'r.ror@example.com', '99853412', 'Sorvei 12B', 'Asker',
'3045', 'Norway'),
('Maciej', 'Hanicki', 'maciek.hanicki@example.com', '98732556', 'Livvei 323C',
'Svarstad', '3275', 'Norway'),
('Kamil', 'Polski', 'kami.pol@example.com', '09854376', 'Lekia 6', 'Sande', '3003',
'Norway'),
('Nina', 'Jensen', 'nina.jensen@example.com', '43498556', 'Trolltunga 34',
'Trondheim', '7010', 'Norway');

-- Table 'Sales'.
INSERT INTO Sales (CustomerID, VIN, EmployeeID, SaleDate, SalePrice)
VALUES
(1, 'VIN12345', 1, '2024-01-15', 300000),
(2, 'VIN12346', 1, '2024-01-20', 500000),
(3, 'VIN12347', 2, '2024-01-25', 200000),
(4, 'VIN12350', 2, '2024-02-10', 450000),
(5, 'VIN12351', 2, '2024-02-15', 350000),
(6, 'VIN12352', 2, '2024-02-20', 550000),
(7, 'VIN12353', 3, '2024-03-01', 250000);

-- Table 'Services'
INSERT INTO Services (VIN, ServiceDate, Description, Cost)
VALUES
('VIN12354', '2024-02-05', 'Technical inspection', 10000),
('VIN12355', '2024-02-10', 'Engine repair', 15000);
```

```sql
-- Table 'Inventories'
INSERT INTO Inventories (VIN, Status, AcquisitionDate, AcquisitionPrice, Price)
VALUES
('VIN12345', 'Sold', '2023-12-01', 250000, 300000),
('VIN12346', 'Sold', '2023-12-05', 450000, 500000),
('VIN12347', 'Sold', '2023-11-15', 150000, 200000),
('VIN12350', 'Sold', '2023-12-10', 400000, 450000),
('VIN12351', 'Sold', '2023-12-20', 300000, 350000),
('VIN12352', 'Sold', '2024-01-05', 500000, 550000),
('VIN12353', 'Sold', '2024-01-10', 200000, 250000),
('VIN12354', 'In Repair', '2023-11-20', 170000, 180000),
('VIN12355', 'In Repair', '2023-12-01', 220000, 230000),
('VIN12348', 'Available', '2023-11-25', 180000, 190000),
('VIN12349', 'Available', '2023-12-15', 210000, 220000),
('VIN12356', 'Available', '2023-12-30', 250000, 260000),
('VIN12357', 'Available', '2024-01-15', 270000, 280000),
('VIN12358', 'Available', '2024-02-01', 290000, 300000),
('VIN12359', 'Available', '2024-02-15', 310000, 320000);
```

ER Diagram:



Deployment Guide: Car_dealership Database

I. Setting up Microsoft SQL Server

1. Download the Microsoft SQL Server installation files from the official Microsoft website.
2. Run the SQL Server setup wizard and follow the on-screen instructions to install SQL Server.
3. During installation, configure the SQL Server instance according to your requirements, including specifying authentication mode, server collation, and instance name.

II. Creating the Database

1. Open SQL Server Management Studio (SSMS).
2. Connect to your SQL Server instance.
3. Execute the SQL script containing the `Car_dealership` database schema. You can do this by opening the SQL script file in SSMS and clicking the "Execute" button, or by copying and pasting the SQL commands into a new query window and executing them.

Example:

```
-- Execute SQL script to create the Car_dealership database schema
USE master;
GO
-- Paste the SQL script here
```

III. Configuring User Permissions (Optional)

1. Create database users for accessing the `Car_dealership` database. This can be done using SQL Server Management Studio or SQL scripts.
2. Assign appropriate roles and permissions to the database users to control access to the database objects.

Example:

```
-- Create a database user and assign permissions
USE Car_dealership;
GO
CREATE USER [username] FOR LOGIN [loginname];
GO
-- Assign roles and permissions
```

IV. Importing Sample Data (Optional)

To use the existing SQL scripts for generating sample data:

1. Select the SQL text you want to copy by dragging your mouse cursor over the content or using the text selection tool in your PDF viewer.

2. Copy the selected SQL text to your clipboard by right-clicking and choosing "Copy" or using the keyboard shortcut (Ctrl + C on Windows, Command + C on Mac).

3. Open your database management tool, such as SQL Server Management Studio (SSMS) or any other tool you prefer.

4. Open a new query window or script editor in your database management tool.

5. Paste the copied SQL text into the query window or script editor by right-clicking and choosing "Paste" or using the keyboard shortcut (Ctrl + V on Windows, Command + V on Mac).

6. Review the pasted SQL code to ensure it copied correctly, then execute the script to generate the sample data in your database.

V. Testing the Database

1. Verify that the `Car_dealership` database is accessible and functioning correctly.
2. Run sample queries to ensure data retrieval and manipulation operations work as expected.
3. Validate data integrity by checking for any inconsistencies or errors.

VI. Backup and Recovery

1. Set up regular database backups to protect against data loss in the event of hardware failures, human errors, or other unforeseen circumstances.
2. Configure backup schedules and retention policies based on your organization's requirements.
3. Familiarize yourself with the backup and recovery procedures in Microsoft SQL Server.