فاز سوم

اعضای گروه:

محمدمهدی احمدی

حوری دهش

سارا سلطانی گردفرامرزی

موضوع پروژه:

سامانه مدیریت کارواش

برای ران شدن پروژه به ترتیب فایل های زیر را ران کنید:

create_tables

functions

procedures

triggers

insert

views

create_tables

۱. جدول Customers:

```
-- Table 1: Customers Table
CREATE TABLE Customers (
    CustomerID INT IDENTITY(1,1) PRIMARY KEY,
    Name NVARCHAR(255),
    Phone NVARCHAR(20),
    Email NVARCHAR(255),
    Address NVARCHAR(255)
);
```

CustomerID: یک ID یکتا برای هر مشتری است که به صورت خودکار افزایش می یابد (Identity) و این کلید اصلی جدول می باشد

Name: نام مشتری

Phone: شماره تلفن مشتری

Email: آدرس ایمیل مشتری

Address: آدرس مشتری

۲. جدول Vehicles:

```
-- Table 2: Vehicles Table
CREATE TABLE Vehicles (
    VehicleID INT IDENTITY(1,1) PRIMARY KEY,
    LicensePlate NVARCHAR(20),
    Model NVARCHAR(255),
    VehicleType NVARCHAR(50),
    PricingFactor NUMERIC(5, 2)
);
```

VehicleID: یک ID یکتا برای هر خودرو است که به صورت خودکار افزایش می یابد (Identity) و این کلید اصلی جدول می باشد

LicensePlate: پلاک خودرو

Model: مدل خودرو

VehicleType: نوع خودرو

PricingFactor: یک ضریب قیمتی برای هر خودرو است که متناسب با خودروها متفاوت می باشد

۳. جدول Services:

```
-- Table 3: Services Table
CREATE TABLE Services (
    ServiceID INT IDENTITY(1,1) PRIMARY KEY,
    ServiceName NVARCHAR(255),
    Description TEXT,
    Price NUMERIC(10, 2)
);
```

ServiceID: یک ID یکتا برای هر خدماتی است که ارائه میشود و به صورت خودکار افزایش می یابد (Identity) و این کلید اصلی جدول می باشد

ServiceName: نام آن سرویسی است که ارائه شده است

Description: توضیحات بیشتر درباره سرویس ها یا خدمات

Price: قیمت آن سرویس است

٤. جدول Orders/Appointments:

```
-- Table 4: Orders/Appointments Table
CREATE TABLE Orders (
    OrderID INT IDENTITY(1,1) PRIMARY KEY,
    CustomerID INT FOREIGN KEY REFERENCES Customers(CustomerID),
    VehicleID INT FOREIGN KEY REFERENCES Vehicles(VehicleID),
    ServiceID INT FOREIGN KEY REFERENCES Services(ServiceID),
    DateTime DATETIME,
    Status NVARCHAR(50),
    TransactionID INT FOREIGN KEY REFERENCES PaymentTransactions(TransactionID),
    OrderPrice NUMERIC(10, 2)
);
```

OrderID: یک ID یکتا برای هر سفارش یا قرار ملاقات هر مشتری است که به صورت خودکار افزایش می یابد (Identity) و این کلید اصلی جدول می باشد

CustomerID: کلید خارجی به جدول Customers است

VehicleID: کلید خارجی به جدول Vehicles است

ServiceID: کلید خارجی به جدول Services است

DateTime: تاریخ و زمان سفارش یا قرار ملاقات

Status: وضعیت سفارش (مثلا در حالت pending یا complete است)

TransactionID: کلید خارجی به جدول PaymentTransactions است

OrderPrice: قیمت کل سفارش می باشد

۵. جدول Employees:

```
-- Table 5: Employees Table
CREATE TABLE Employees (
    EmployeeID INT IDENTITY(1,1) PRIMARY KEY,
    Name NVARCHAR(255),
    Phone NVARCHAR(20),
    Email NVARCHAR(255),
    Role NVARCHAR(50)
);
```

EmployeeID: یک ID یکتا برای هر کارمند است که به صورت خودکار افزایش می یابد (Identity)و این کلید اصلی جدول می باشد

Name: نام کارمند

Phone: شماره تلفن کارمند

Email: آدرس ایمیل کارمند

Role: نقش کارمند در کارواش

۶. جدول Payment Transactions:

```
-- Table 6: Payment Transactions Table
CREATE TABLE PaymentTransactions (
    TransactionID INT IDENTITY(1,1) PRIMARY KEY,
    PaymentType NVARCHAR(50),
    Amount NUMERIC(10, 2),
    TransactionDateTime DATETIME
);
```

TransactionID: یک ID یکتا برای هر تراکنش پرداختی است که به صورت خودکار افزایش می یابد (Identity) و این کلید اصلی جدول می باشد

PaymentType: نوع پرداخت است

Amount: مقدار پرداخت

TransactionDateTime: تاریخ و زمان انجام تراکنش پرداختی

۷. جدول Feedback and Reviews:

```
-- Table 7: Feedback and Reviews Table
CREATE TABLE Feedbacks (
    ReviewID INT IDENTITY(1,1) PRIMARY KEY,
    CustomerID INT FOREIGN KEY REFERENCES Customers(CustomerID),
    ServiceID INT FOREIGN KEY REFERENCES Services(ServiceID),
    Rating NUMERIC(4, 2) CHECK (Rating >= 0 AND Rating <= 5),
    Comments TEXT,
    ReviewDateTime DATETIME
);
```

ReviewID: یک ID یکتا برای هر بازخورد و نقد است که به صورت خودکار افزایش می یابد (Identity) و این کلید اصلی جدول می باشد

CustomerID: کلید خارجی به جدول Customers است

ServiceID: کلید خارجی به جدول Services است

Rating: امتیاز داده شده برای آن خدماتی که گرفته اند

Comments: نظرات و توضیحات اضافی

ReviewDateTime: تاریخ و زمان ارسال بازخورد

functions

۱. CalculateOrderPrice function:

```
--CalculateOrderPrice function
CREATE FUNCTION CalculateOrderPrice (
    @ServicePrice NUMERIC(10, 2),
    @PricingFactor NUMERIC(5, 2)
)
RETURNS NUMERIC(10, 2)
AS
BEGIN
    DECLARE @OrderPrice NUMERIC(10, 2);
    SET @OrderPrice = @ServicePrice * @PricingFactor;
    RETURN @OrderPrice;
END;
GO
```

از تابع CalculateOrderPrice برای محاسبه قیمت نهایی یک سفارش، بر اساس قیمت سرویسی که استفاده کرده و فاکتور قیمت ارائه شده، استفاده میشود.

ورودی ها:

ServicePrice: یک عدد اعشاری با دقت ۱۰ رقم در کل و ۲ رقم در ممیز، که نمایانگر قیمت خدمت مورد نظر در جدول Services است.

PricingFactor: یک عدد اعشاری با دقت ۵ رقم در کل و ۲ رقم در ممیز، که نمایانگر فاکتور قیمت مورد استفاده برای محاسبه قیمت سفارش است.

عملیات:

تابع ابتدا یک متغیر محلی به نام OrderPrice ایجاد می کند که برای ذخیره قیمت نهایی سفارش استفاده می شود.

سپس با استفاده از فرمول ServicePrice * PricingFactor، قیمت سفارش محاسبه می شود.

خروجی:

تابع مقدار محاسبه شده برای قیمت سفارش را به عنوان خروجی باز می گرداند یعنی OrderPrice را که این مقدار یک عدد اعشاری با دقت ۱۰ رقم در کل و ۲ رقم در ممیز است.

۲. GetAverageRatingForService function:

```sql
--GetAverageRatingForService function
CREATE FUNCTION GetAverageRatingForService (@ServiceID INT)
RETURNS FLOAT
AS
BEGIN
    DECLARE @AverageRating FLOAT;

    SELECT @AverageRating = AVG(Rating)
    FROM Feedbacks
    WHERE ServiceID = @ServiceID;

    RETURN ISNULL(@AverageRating, 0);
END;
GO
```

از تابع GetAverageRatingForService برای محاسبه میانگین امتیازها برای یک سرویس مشخص استفاده می شود.

ورودی:

ServiceID: یک عدد صحیح که نشان دهنده ID یک سرویس در جدول Services است.

عملیات:

تابع ابتدا یک متغیر محلی به نام AverageRating ایجاد می کند که برای ذخیره میانگین امتیازها استفاده می شود.

سپس با استفاده از دستور

SELECT @AverageRating = AVG(Rating) FROM Feedbacks

WHERE ServiceID = @ServiceID

میانگین امتیازها برای سرویس مشخص شده در ورودی محاسبه می شود.

خروجی:

تابع مقدار محاسبه شده برای میانگین امتیازها را به عنوان خروجی باز می گرداند و اگر هیچ امتیازی برای سرویس مورد نظر وجود نداشته باشد یعنی سرویس امتیازی دریافت نکرده باشد از تابع ISNULL(@AverageRating, ۰) استفاده میشود که یک مقدار پیش فرض صفر را برمی گرداند.

۳. GetTotalSalesByService function:

```
--GetTotalSalesByService function
CREATE FUNCTION GetTotalSalesByService
    (@ServiceID INT)
RETURNS NUMERIC(10, 2)
AS
BEGIN
    DECLARE @TotalSales NUMERIC(10, 2);

    SELECT @TotalSales = SUM(OrderPrice)
    FROM Orders
    WHERE ServiceID = @ServiceID;

    RETURN ISNULL(@TotalSales, 0);
END;
GO
```

از تابع GetTotalSalesByService برای محاسبه کل فروش بر اساس یک سرویس مشخص شده استفاده میشود.

ورودی:

ServiceID: یک عدد صحیح که نشان دهنده ID یک سرویس در جدول Services است.

عملیات:

تابع ابتدا یک متغیر محلی به نام TotalSales ایجاد می کند که برای ذخیره مجموع فروش استفاده می شود.

سپس با استفاده از دستور

SELECT @TotalSales = SUM(OrderPrice) FROM Orders

WHERE ServiceID = @ServiceID

مجموع فروش برای سرویس مشخص شده در ورودی محاسبه می شود.

خروجی:

تابع مقدار محاسبه شده برای مجموع فروش را به عنوان خروجی باز می گرداند و اگر هیچ فروشی برای سرویس مورد نظر وجود نداشته باشد یعنی هیچ سفارشی برای سرویس مورد نظر در جدول Orders وجود نداشته باشد، از تابع ISNULL(@TotalSales, ۰) استفاده میشود که یک مقدار پیش فرض صفر را برمی گرداند.

٤. GetServiceSummary function:

```
--GetServiceSummary function
CREATE FUNCTION GetServiceSummary()
RETURNS TABLE
AS
RETURN (
    SELECT
        S.ServiceID,
        S.ServiceName,
        COUNT(F.ReviewID) AS NumberOfReviews,
        dbo.GetAverageRatingForService(S.ServiceID) AS AverageServiceRating,
        dbo.GetTotalSalesByService(S.ServiceID) AS TotalSales
    FROM
        Services AS S
        LEFT JOIN Feedbacks AS F ON S.ServiceID = F.ServiceID
    GROUP BY
        S.ServiceID, S.ServiceName
);
GO
```

از تابع GetServiceSummary برای دریافت خلاصه‌ای از اطلاعات مرتبط با خدمات شامل تعداد نظرات، میانگین امتیازات و کل فروش استفاده میشود.

تابع GetServiceSummary یک جدول خروجی ایجاد می کند که شامل ستون‌های زیر است:

ServiceID: ID سرویس از جدول Services

ServiceName: نام سرویس از جدول Services

NumberOfReviews: تعداد نظرات مرتبط با سرویس از جدول Feedbacks

AverageServiceRating: میانگین امتیازهای مرتبط با سرویس که با استفاده از تابع GetAverageRatingForService محاسبه می شود.

TotalSales: کل فروش مرتبط با سرویس که با استفاده از تابع GetTotalSalesByService محاسبه میشود.

عملیات:

در داخل تابع یک جدول موقت به نام S برای جدول Services و یک جدول موقت به نام F برای جدول Feedbacks تعریف می شود. سپس از یک LEFT JOIN بین جدول Services و جدول Feedbacks با شرط ارتباط بین ServiceID در هر دو جدول استفاده می شود در نهایت با group by کردن بر اساس ستون های ServiceID و ServiceName در جدول Services، تعداد نظرات، میانگین امتیازات و کل فروش برای هر سرویس محاسبه می شود.

و در آخر این تابع یک جدول به عنوان خروجی برمی گرداند.

procedures

۱. AddNewOrder procedure:

```
--AddNewOrder procedure
CREATE PROCEDURE AddNewOrder (
    @CustomerID INT,
    @VehicleID INT,
    @ServiceID INT,
    @NewOrderID INT OUTPUT
)
AS
BEGIN
    DECLARE @ServicePrice NUMERIC(10, 2);
    DECLARE @PricingFactor NUMERIC(5, 2);
    DECLARE @OrderPrice NUMERIC(10, 2);

    -- Get service price and vehicle pricing factor
    SELECT @ServicePrice = Price, @PricingFactor = PricingFactor
    FROM Services s
    JOIN Vehicles v ON v.VehicleID = @VehicleID
    WHERE s.ServiceID = @ServiceID;

    -- Calculate order price
    SET @OrderPrice = dbo.CalculateOrderPrice(@ServicePrice, @PricingFactor);

    -- Insert to Orders table
    INSERT INTO Orders (CustomerID, VehicleID, ServiceID, DateTime, Status, TransactionID, OrderPrice)
    VALUES (@CustomerID, @VehicleID, @ServiceID, GETDATE(), 'pending', NULL, @OrderPrice);

    -- Get the newly inserted OrderID
    SET @NewOrderID = SCOPE_IDENTITY();
END;
GO
```

از پروسیجر AddNewOrder برای افزودن یک سفارش جدید به جدول Orders استفاده میشود.

ورودی ها:

CustomerID: ID مشتری که سفارش را ثبت می کند

VehicleID: ID وسیله نقلیه مرتبط با سفارش

ServiceID: ID سرویس مرتبط با سفارش

NewOrderID: یک پارامتر OUTPUT که برای دریافت ID سفارش جدید بعد از ثبت استفاده می شود

عملیات:

ابتدا متغیرهای محلی ServicePrice و PricingFactor برای ذخیره قیمت سرویس و فاکتور قیمت وسیله نقلیه تعریف می شوند سپس اطلاعات این دو تا متغییر از جدول Services و Vehicles با استفاده از یک JOIN به دست می آیند.

با فراخوانی تابع CalculateOrderPrice قیمت سفارش یعنی OrderPrice براساس قیمت سرویس و فاکتور قیمت وسیله نقلیه محاسبه می شود.

در آخر یک رکورد جدید به جدول Orders اضافه می شود که این رکورد شامل اطلاعات مشتری، وسیله نقلیه، سرویس، تاریخ و وضعیت سفارش و قیمت سفارش است سپس با استفاده از SCOPE_IDENTITY، شناسه سفارش جدید به عنوان مقدار پارامتر NewOrderID تنظیم می شود. این مقدار به کاربر ارائه میشود تا بتواند شناسه سفارش جدید را دریافت کند.

۲. ProcessOrderPayments procedure:

```sql
--ProcessOrderPayments procedure
CREATE PROCEDURE ProcessOrderPayments
    @OrderIDs NVARCHAR(MAX),
    @PaymentType NVARCHAR(50)
AS
BEGIN
    DECLARE @PaymentID INT;
    DECLARE @TotalOrderPrice NUMERIC(10, 2);

    -- Create Payment entry
    INSERT INTO PaymentTransactions (PaymentType, Amount, TransactionDateTime)
    VALUES (@PaymentType, 0, GETDATE());

    -- Get the newly created Payment ID
    SET @PaymentID = SCOPE_IDENTITY();

    -- Update TransactionID for each order in the list
    UPDATE Orders
    SET TransactionID = @PaymentID
    WHERE OrderID IN (SELECT CAST(value AS INT) FROM STRING_SPLIT(@OrderIDs, ','));

    -- Calculate the total order price
    SELECT @TotalOrderPrice = ISNULL(SUM(OrderPrice), 0)
    FROM Orders
    WHERE OrderID IN (SELECT CAST(value AS INT) FROM STRING_SPLIT(@OrderIDs, ','));

    -- Update the Payment amount with the total order price
    UPDATE PaymentTransactions
    SET Amount = @TotalOrderPrice
    WHERE TransactionID = @PaymentID;
END;
GO
```

از پروسیجر ProcessOrderPayments برای پردازش پرداخت های مرتبط با یک یا چند سفارش، ایجاد تراکنش پرداخت جدید و به روزرسانی اطلاعات مربوط به پرداخت استفاده میشود.

ورودی ها:

OrderIDs: یک رشته است که حاوی IDهای سفارشات جدا شده با ویرگول است. این IDها با استفاده از تابع STRING_SPLIT از رشته جدا می شوند و به صورت لیست به پروسیجر وارد می شوند.

PaymentType: نوع پرداخت مورد نظر برای ثبت در جدول PaymentTransactions است.

عملیات:

ابتدا یک رکورد جدید برای پرداخت در جدول PaymentTransactions ایجاد میشود. مقدار پیش فرض برای Amount به صورت صفر است و با استفاده از تابع GETDATE تاریخ و زمان فعلی ثبت میشود.

با استفاده از SCOPE_IDENTITY، شناسه پرداخت جدید به متغیر PaymentID تنظیم می شود. همچنین با استفاده از UPDATE، شناسه تراکنش برای هر سفارش موجود در لیست OrderIDs در جدول Orders به ID پرداخت جدید تنظیم می شود.

سپس با استفاده از دستور

```
SELECT @TotalOrderPrice = ISNULL(SUM(OrderPrice), ۰) FROM Orders

WHERE OrderID IN (SELECT CAST(value AS INT) FROM STRING_SPLIT(@OrderIDs, ','))
```

مجموع قیمت سفارشات مرتبط با پرداخت محاسبه می شود.

در نهایت با استفاده از UPDATE دیگری، مقدار Amount در رکورد مربوط به پرداخت در جدول PaymentTransactions به مقدار مجموع قیمت سفارشات تنظیم می شود.

۳. AddNewCustomer procedure:

```
--AddNewCustomer procedure
CREATE PROCEDURE AddNewCustomer
    @Name NVARCHAR(255),
    @Phone NVARCHAR(20),
    @Email NVARCHAR(255),
    @Address NVARCHAR(255),
    @EmployeeRole NVARCHAR(50)
AS
BEGIN
    IF @EmployeeRole = 'Cashier' OR @EmployeeRole = 'Admin'
    BEGIN
        INSERT INTO Customers (Name, Phone, Email, Address)
        VALUES (@Name, @Phone, @Email, @Address);
    END
    ELSE
    BEGIN
        RAISERROR('Only Cashier or Admin can add new customers.', 16, 1);
    END
END;
GO
```

از پروسیجر AddNewCustomer برای افزودن مشتری جدید به جدول Customers استفاده می شود.

ورودی ها:

Name: نام مشتری جدید

Phone: شماره تلفن مشتری جدید

Email: آدرس ایمیل مشتری جدید

Address: آدرس مشتری جدید

EmployeeRole: نقش کارمند

عملیات:

ابتدا با استفاده از یک شرط IF بررسی می شود که نقش کارمندی که این عملیات را انجام می‌دهد، Cashier یا Admin است یا هیچکدام. اگر کارمند نقش Cashier یا Admin را داشته باشد یک رکورد جدید به جدول Customers اضافه میشود با اطلاعات مشتری جدید که از ورودی های پروسیجر گرفته شده‌اند و در صورتی که نقش کارمند نامعتبر باشد یعنی نه Cashier و نه Admin باشد، با اجرای RAISEERROR یک پیام خطا ایجاد می شود و اجرای پروسیجر متوقف می شود.

٤. AddNewVehicle procedure:

```
--AddNewVehicle procedure
CREATE PROCEDURE AddNewVehicle
    @LicensePlate NVARCHAR(20),
    @Model NVARCHAR(255),
    @VehicleType NVARCHAR(50),
    @PricingFactor NUMERIC(5, 2),
    @EmployeeRole NVARCHAR(50)
AS
BEGIN
    IF @EmployeeRole = 'Cashier' OR @EmployeeRole = 'Admin'
    BEGIN
        INSERT INTO Vehicles (LicensePlate, Model, VehicleType, PricingFactor)
        VALUES (@LicensePlate, @Model, @VehicleType, @PricingFactor);
    END
    ELSE
    BEGIN
        RAISERROR('Only Cashier or Admin can add new vehicles.', 16, 1);
    END
END;
GO
```

از پروسیجر AddNewVehicle برای افزودن وسیله نقلیه جدید به جدول Vehicles استفاده می شود.

ورودی ها:

LicensePlate: پلاک وسیله نقلیه جدید

Model: مدل وسیله نقلیه جدید

VehicleType: نوع وسیله نقلیه جدید

PricingFactor: فاکتور قیمت برای وسیله نقلیه جدید

EmployeeRole: نقش کارمند

عملیات:

ابتدا با استفاده از یک شرط IF بررسی می شود که نقش کارمندی که این عملیات را انجام می‌دهد، Cashier یا Admin است یا هیچکدام. اگر کارمند نقش Cashier یا Admin را داشته باشد یک رکورد جدید به جدول Vehicles اضافه میشود با اطلاعات وسیله نقلیه جدید که از ورودی های پروسیجر گرفته شده‌اند و در صورتی که نقش کارمند نامعتبر باشد یعنی نه Cashier و نه Admin باشد، با اجرای RAISEERROR یک پیام خطا ایجاد می شود و اجرای پروسیجر متوقف می شود.

o. AddNewService procedure:

```
--AddNewService procedure
CREATE PROCEDURE AddNewService
    @ServiceName NVARCHAR(255),
    @Description TEXT,
    @Price NUMERIC(10, 2),
    @EmployeeRole NVARCHAR(50)
AS
BEGIN
    IF @EmployeeRole = 'Cashier' OR @EmployeeRole = 'Admin'
    BEGIN
        INSERT INTO Services (ServiceName, Description, Price)
        VALUES (@ServiceName, @Description, @Price);
    END
    ELSE
    BEGIN
        RAISERROR('Only Cashier or Admin can add new services.', 16, 1);
    END
END;
GO
```

از پروسیجر AddNewService برای افزودن سرویس جدید به جدول Services استفاده می‌شود.

ورودی ها:

ServiceName: نام سرویس جدید

Description: توضیحات سرویس جدید

Price: قیمت سرویس جدید

EmployeeRole: نقش کارمند

عملیات:

ابتدا با استفاده از یک شرط IF بررسی می شود که نقش کارمندی که این عملیات را انجام می‌دهد، Cashier یا Admin است یا هیچکدام. اگر کارمند نقش Cashier یا Admin را داشته باشد یک رکورد جدید به جدول Services اضافه میشود با اطلاعات سرویس جدید که از ورودی های پروسیجر گرفته شده‌اند و در صورتی که

نقش کارمند نامعتبر باشد یعنی نه Cashier و نه Admin باشد، با اجرای RAISEERROR یک پیام خطا ایجاد می شود و اجرای پروسیجر متوقف می شود.

۶. AddNewFeedback procedure:

```
--AddNewFeedback procedure
CREATE PROCEDURE AddNewFeedback
    @CustomerID INT,
    @ServiceID INT,
    @Rating NUMERIC(4, 2),
    @Comments TEXT,
    @EmployeeRole NVARCHAR(50)
AS
BEGIN
    IF @EmployeeRole = 'Cashier' OR @EmployeeRole = 'Admin'
    BEGIN
        INSERT INTO Feedbacks (CustomerID, ServiceID, Rating, Comments, ReviewDateTime)
        VALUES (@CustomerID, @ServiceID, @Rating, @Comments, GETDATE());
    END
    ELSE
    BEGIN
        RAISERROR('Only Cashier or Admin can add new feedbacks.', 16, 1);
    END
END;
GO
```

از پروسیجر AddNewFeedback برای افزودن بازخورد جدید به جدول Feedbacks استفاده می شود.

ورودی ها:

CustomerID: ID مشتری که بازخورد را ثبت می کند

ServiceID: ID سرویس مورد بازخورد

Rating: امتیاز اختصاص داده شده به سرویس

Comments: نظرات یا بازخورد مشتری

EmployeeRole: نقش کارمند

عملیات:

ابتدا با استفاده از یک شرط IF بررسی می شود که نقش کارمندی که این عملیات را انجام می‌دهد، Cashier یا Admin است یا هیچکدام. اگر کارمند نقش Cashier یا Admin را داشته باشد یک رکورد جدید به جدول Feedbacks اضافه میشود با اطلاعات بازخورد که از ورودی های پروسیجر گرفته شده‌اند. همچنین تاریخ و زمان ثبت بازخورد با استفاده از GETDATE تنظیم می شود. در صورتی که نقش کارمند نامعتبر باشد یعنی نه Cashier و نه Admin باشد، با اجرای RAISEERROR یک پیام خطا ایجاد می شود و اجرای پروسیجر متوقف می شود.

۷. AddNewEmployee procedure:

```
--AddNewEmployee procedure
CREATE PROCEDURE AddNewEmployee
    @Name NVARCHAR(255),
    @Phone NVARCHAR(20),
    @Email NVARCHAR(255),
    @Role NVARCHAR(50),
    @AdminRole NVARCHAR(50)
AS
BEGIN
    IF @AdminRole = 'Admin'
    BEGIN
        INSERT INTO Employees (Name, Phone, Email, Role)
        VALUES (@Name, @Phone, @Email, @Role);
    END
    ELSE
    BEGIN
        RAISERROR('Only Admin can add new employees.', 16, 1);
    END
END;
GO
```

از پروسیجر AddNewEmployee برای افزودن کارمند جدید به جدول Employees استفاده می شود.

ورودی ها:

Name: نام کارمند جدید

Phone: شماره تلفن کارمند جدید

Email: آدرس ایمیل کارمند جدید

Role: نقش کارمند جدید

AdminRole: نقش مدیر سیستم که عملیات اضافه کردن کارمند را انجام می‌دهد

عملیات:

ابتدا با استفاده از یک شرط IF بررسی می شود که نقش مدیر سیستمی که این عملیات را انجام می‌دهد، Admin است یا نه. اگر مدیر سیستم نقش Admin را داشته باشد یک رکورد جدید به جدول Employees اضافه میشود با اطلاعات کارمند جدید که از ورودی های پروسیجر گرفته شده‌اند و در صورتی که نقش مدیر سیستم نامعتبر باشد یعنی Admin نباشد، با اجرای RAISEERROR یک پیام خطا ایجاد می شود و اجرای پروسیجر متوقف می شود.

triggers

۱. tr_UpdateOrderStatusOnPayment trigger :

```
-- Create the tr_UpdateOrderStatusOnPayment trigger
CREATE TRIGGER tr_UpdateOrderStatusOnPayment
ON Orders
AFTER UPDATE
AS
BEGIN
    IF UPDATE(TransactionID)
    BEGIN
        UPDATE Orders
        SET Status = 'complete'
        WHERE OrderID IN (SELECT OrderID FROM INSERTED);
    END
END;
GO
```

عملیات: به طور خلاصه، این تریگر به‌طور خودکار «وضعیت» سفارش‌ها را هنگامی که ستون «TransactionID»آپدیت می‌شود، به مقدار «complete » تغییرمیدهد .فرض بر این است که به‌روزرسانی «TransactionID»به معنای تکمیل تراکنش یا پرداخت مربوط به سفارش است .

این عملیات از جدول INSERTED استفاده می کند که یک جدول ویژه در SQL Server است که کپی هایی از ردیف های آپدیت شده یا تغییر یافته را در طول عملیات به روز رسانی نگه می دارد.

۲. tr_UpdatePendingOrdersOnServicePriceChange trigger:

```
--Create the tr_UpdatePendingOrdersOnServicePriceChange trigger
CREATE TRIGGER tr_UpdatePendingOrdersOnServicePriceChange
ON Services
AFTER UPDATE
AS
BEGIN
    IF UPDATE(Price)
    BEGIN
        UPDATE o
        SET o.OrderPrice = dbo.CalculateOrderPrice(i.Price, v.PricingFactor)
        FROM Orders o
        INNER JOIN INSERTED i ON o.ServiceID = i.ServiceID
        INNER JOIN Vehicles v ON v.VehicleID = o.VehicleID
        WHERE o.Status = 'pending';
    END
END;
GO
```

عملیات: این تریگر روی جدول "Services" ایجاد می شود .این تریگر طوری طراحی شده است که پس از انجام عملیات به روز رسانی در جدول "سرویس ها"، به ویژه زمانی که ستون "قیمت" به روز می شود، اجرا شود. به طور خلاصه، این تریگرهنگام به‌روزرسانی قیمت یک سرویس، «OrderPrice»را در جدول «Orders» برای سفارش‌هایی با وضعیت «در انتظار» به‌روزرسانی می‌کند .قیمت سفارش جدید با استفاده از تابع CalculateOrderPrice محاسبه می شود که قیمت خدمات جدید و فاکتور قیمت گذاری وسیله نقلیه مرتبط را در نظر می گیرد.

۳. tr_UpdateOrderPriceOnVehicleChange trigger:

```
--Create the tr_UpdateOrderPriceOnVehicleChange trigger
CREATE TRIGGER tr_UpdateOrderPriceOnVehicleChange
ON Orders
AFTER UPDATE
AS
BEGIN
    IF UPDATE(VehicleID) OR UPDATE(ServiceID)
    BEGIN
        UPDATE o
        SET OrderPrice = dbo.CalculateOrderPrice(s.Price, v.PricingFactor)
        FROM Orders o
        INNER JOIN INSERTED i ON o.OrderID = i.OrderID
        INNER JOIN Services s ON s.ServiceID = i.ServiceID
        INNER JOIN Vehicles v ON v.VehicleID = i.VehicleID
        WHERE o.Status = 'pending';
    END
END;
GO
```

عملیات: به طور خلاصه، زمانی که ستون‌های «VehicleID» یا «ServiceID» به‌روزرسانی می‌شوند، این تریگر «OrderPrice» را در جدول «Orders» برای سفارش‌هایی با وضعیت «در انتظار» به‌روزرسانی می‌کند. قیمت سفارش جدید با استفاده از یک تابع تعریف شده محاسبه می شود که قیمت خدمات و فاکتور قیمت گذاری وسیله نقلیه مرتبط را در نظر می گیرد.

Inserts

۱. اضافه کردن دیتا به جدول customer:

```
-- Insert data into Customers Table
EXEC AddNewCustomer 'John Doe', '123-456-7890', 'john@example.com', '123 Main St', 'Cashier';
EXEC AddNewCustomer 'Jane Smith', '987-654-3210', 'jane@example.com', '456 Oak Ave', 'Cashier';
EXEC AddNewCustomer 'Bob Johnson', '555-123-4567', 'bob@example.com', '789 Elm St', 'Cashier';
EXEC AddNewCustomer 'Alice Williams', '222-333-4444', 'alice@example.com', '101 Pine Ave', 'Cashier';
EXEC AddNewCustomer 'Charlie Brown', '888-999-0000', 'charlie@example.com', '202 Maple Dr', 'Cashier';
EXEC AddNewCustomer 'Eva Davis', '777-666-5555', 'eva@example.com', '303 Birch Ln', 'Cashier';
EXEC AddNewCustomer 'Frank White', '444-777-8888', 'frank@example.com', '404 Cedar Rd', 'Cashier';
EXEC AddNewCustomer 'Grace Turner', '999-111-2222', 'grace@example.com', '505 Oak St', 'Cashier';
EXEC AddNewCustomer 'Harry Black', '666-444-3333', 'harry@example.com', '606 Pine Ln', 'Cashier';
EXEC AddNewCustomer 'Ivy Green', '111-222-3333', 'ivy@example.com', '707 Maple Rd', 'Cashier';
GO
```

۲. اضافه کردن دیتا به جدول vehicles:

```
-- Insert data into Vehicles Table
EXEC AddNewVehicle 'ABC123', 'Toyota Camry', 'Sedan', 1.00, 'Cashier';
EXEC AddNewVehicle 'XYZ789', 'Honda Accord', 'Coupe', 1.20, 'Cashier';
EXEC AddNewVehicle '123XYZ', 'Ford Explorer', 'SUV', 1.10, 'Cashier';
EXEC AddNewVehicle '789ABC', 'Chevrolet Malibu', 'Sedan', 1.00, 'Cashier';
EXEC AddNewVehicle '456DEF', 'Nissan Rogue', 'SUV', 1.10, 'Cashier';
EXEC AddNewVehicle 'JKL321', 'Toyota Prius', 'Hatchback', 1.30, 'Cashier';
EXEC AddNewVehicle 'MNO987', 'Jeep Wrangler', 'SUV', 1.10, 'Cashier';
EXEC AddNewVehicle 'DEF456', 'Ford Mustang', 'Convertible', 1.40, 'Cashier';
EXEC AddNewVehicle 'GHI654', 'Honda CR-V', 'SUV', 1.10, 'Cashier';
EXEC AddNewVehicle 'PQR321', 'Chevrolet Silverado', 'Truck', 3.00, 'Cashier';
GO
```

۳. اضافه کردن دیتا به جدول Services:

```
-- Insert data into Services Table
EXEC AddNewService 'Outside Car Wash', 'Exterior cleaning of the car', 15.00, 'Cashier';
EXEC AddNewService 'Inside Car Wash', 'Interior cleaning of the car', 20.00, 'Cashier';
EXEC AddNewService 'Full Car Detailing', 'Comprehensive interior and exterior cleaning', 32.00, 'Cashier';
EXEC AddNewService 'Tire Rotation', 'Rotation of vehicle tires', 10.00, 'Cashier';
EXEC AddNewService 'Oil Change', 'Engine oil replacement', 30.00, 'Cashier';
GO
```

٤. اضافه کردن دیتا به جدول Employees:

```
-- Insert data into Employees Table
EXEC AddNewEmployee 'Caroline Davis', '111-222-3333', 'caroline@example.com', 'Car Washer', 'Admin';
EXEC AddNewEmployee 'David Smith', '444-555-6666', 'david@example.com', 'Cashier', 'Admin';
EXEC AddNewEmployee 'Emma Brown', '777-888-9999', 'emma@example.com', 'Car Washer', 'Admin';
EXEC AddNewEmployee 'George White', '333-444-5555', 'george@example.com', 'Cashier', 'Admin';
EXEC AddNewEmployee 'Hannah Black', '666-777-8888', 'hannah@example.com', 'Car Washer', 'Admin';
EXEC AddNewEmployee 'Admin', '888-444-5555', 'admin@example.com', 'Admin', 'Admin';
GO
```

۵. اضافه کردن دیتا به جدول Feedbacks:

```
-- Insert data into Feedbacks Table
EXEC AddNewFeedback 1, 1, 5, 'Great service!', 'Cashier';
EXEC AddNewFeedback 2, 2, 4, 'Prompt and efficient', 'Cashier';
EXEC AddNewFeedback 3, 3, 5, 'Excellent detailing work', 'Cashier';
EXEC AddNewFeedback 4, 4, 3, 'Average service, could improve', 'Cashier';
EXEC AddNewFeedback 5, 4, 4, 'Good services', 'Cashier';
EXEC AddNewFeedback 6, 5, 4, 'Good oil change service', 'Cashier';
GO
```

٦. اضافه کردن دیتا به جدول orders , PaymentTransactions:

عملیات: ابتدا سه خط اولیه سه متغیر را برای ذخیره مقادیر خروجی پراسیجر ذخیره شده تعریف میکنند. سپس سه خط بعدی پراسیجر ذخیره شده AddNewOrder را سه بار با پارامترهای مختلف اجرا می کنند. این پراسیجر کارش افزودن سفارشات جدید است. مثلا در اینسرت اولی پارامترها شامل ۱ برای ایدی مشتری، ۱ برای ایدی وسیله نقلیه، و ایدی های خدمات مختلف (۱، ۲، و ۵) است. ایدی های سفارشات تازه درج شده در پارامترهای خروجی ثبت می شوند. سپس سه ایدی سفارشات تازه درج شده را به هم متصل کرده و در متغیر orderIDS۱ ریخته میشود. سپس پراسیجر دیگری را اجرا می کند (ProcessOrderPayments) که این پراسیجر بالاتر توضیح داده شد.

```
-- Insert data to Orders and PaymentTransactions
DECLARE @NewlyInsertedOrderID1 INT;
DECLARE @NewlyInsertedOrderID2 INT;
DECLARE @NewlyInsertedOrderID3 INT;
EXEC AddNewOrder 1, 1, 1, @NewlyInsertedOrderID1 OUTPUT;
EXEC AddNewOrder 1, 1, 2, @NewlyInsertedOrderID2 OUTPUT;
EXEC AddNewOrder 1, 1, 5, @NewlyInsertedOrderID3 OUTPUT;
DECLARE @OrderIDs1 NVARCHAR(MAX);
SET @OrderIDs1 = CONCAT(@NewlyInsertedOrderID1, ',', @NewlyInsertedOrderID2, ',', @NewlyInsertedOrderID3);
EXEC ProcessOrderPayments @OrderIDs1, 'Cash';
GO

DECLARE @NewlyInsertedOrderID4 INT;
DECLARE @NewlyInsertedOrderID5 INT;
DECLARE @NewlyInsertedOrderID6 INT;
EXEC AddNewOrder 2, 10, 1, @NewlyInsertedOrderID4 OUTPUT;
EXEC AddNewOrder 2, 10, 2, @NewlyInsertedOrderID5 OUTPUT;
EXEC AddNewOrder 2, 10, 4, @NewlyInsertedOrderID6 OUTPUT;
DECLARE @OrderIDs2 NVARCHAR(MAX);
SET @OrderIDs2 = CONCAT(@NewlyInsertedOrderID4, ',', @NewlyInsertedOrderID5, ',', @NewlyInsertedOrderID6);
EXEC ProcessOrderPayments @OrderIDs2, 'Credit Card';
GO
```

```
DECLARE @NewlyInsertedOrderID7 INT;
DECLARE @NewlyInsertedOrderID8 INT;
EXEC AddNewOrder 3, 7, 3, @NewlyInsertedOrderID7 OUTPUT;
EXEC AddNewOrder 3, 7, 5, @NewlyInsertedOrderID8 OUTPUT;
DECLARE @OrderIDs3 NVARCHAR(MAX);
SET @OrderIDs3 = CONCAT(@NewlyInsertedOrderID7, ',', @NewlyInsertedOrderID8);
EXEC ProcessOrderPayments @OrderIDs3, 'Credit Card';
GO

DECLARE @NewlyInsertedOrderID9 INT;
DECLARE @NewlyInsertedOrderID10 INT;
EXEC AddNewOrder 4, 8, 4, @NewlyInsertedOrderID9 OUTPUT;
EXEC AddNewOrder 4, 8, 5, @NewlyInsertedOrderID10 OUTPUT;
DECLARE @OrderIDs4 NVARCHAR(MAX);
SET @OrderIDs4 = CONCAT(@NewlyInsertedOrderID9, ',', @NewlyInsertedOrderID10);
EXEC ProcessOrderPayments @OrderIDs4, 'Cash';
GO
```

Views

۱. CustomerOrderView View:

عملیات: این ویو چندین جدول مرتبط را با یکدیگر جوین کرده تا یک نمای تلفیقی از سفارشات مشتری ارائه دهد. در واقع یک نمای کلی جامع از سفارشات مشتری، از جمله جزئیات مشتری، اطلاعات سفارش، جزئیات تراکنش پرداخت، جزئیات خدمات و جزئیات خودرو را ارائه می دهد.

```
--CustomerOrdersView view
CREATE VIEW CustomerOrdersView AS
SELECT
    C.Name AS CustomerName,
    O.OrderID,
    O.DateTime,
    O.Status,
    O.OrderPrice,
    PT.PaymentType,
    PT.Amount AS TotalAmount,
    PT.TransactionDateTime,
    S.ServiceName,
    S.Description AS ServiceDescription,
    S.Price AS ServicePrice,
    V.LicensePlate,
    V.Model,
    V.VehicleType,
    V.PricingFactor
FROM Customers C
JOIN Orders O ON C.CustomerID = O.CustomerID
JOIN PaymentTransactions PT ON O.TransactionID = PT.TransactionID
JOIN Services S ON O.ServiceID = S.ServiceID
JOIN Vehicles V ON O.VehicleID = V.VehicleID;
GO
```

۲. CustomerPaymentView View:

عملیات: این ویو با پیوستن جداول Customers، Orders و PaymentTransaction اطلاعاتی در مورد پرداخت های مشتری ارائه می دهد. به طور خلاصه، CustomerPaymentsView یک نمای تلفیقی از پرداخت های مشتری ارائه می دهد که نام مشتری را همراه با نوع پرداخت، مبلغ و تاریخ/زمان تراکنش نشان می دهد. این ویو برای حذف نام‌های تکراری مشتریان در مجموعه نتایج طراحی شده است و نمای کلی واضحی از پرداخت‌های انجام شده توسط مشتریان فردی ارائه می‌کند.

```
--CustomerPaymentsView view
CREATE VIEW CustomerPaymentsView AS
SELECT
    DISTINCT C.Name AS CustomerName,
    PT.PaymentType,
    PT.Amount,
    PT.TransactionDateTime
FROM Customers C
JOIN Orders O ON C.CustomerID = O.CustomerID
JOIN PaymentTransactions PT ON O.TransactionID = PT.TransactionID;
GO
```

۳. CustomerFeedbackAndServicesView View:

عملیات: این ویو نمایی است که اطلاعاتی درباره بازخورد مشتری و خدمات مرتبط با آن بازخورد با پیوستن جداول مشتریان، بازخوردها و خدمات ارائه می دهد. این ویو به طور خلاصه، یک نمای تلفیقی از بازخورد مشتری، از جمله نام مشتری، رتبه بندی داده شده، نظرات ارائه شده و نام سرویس مرتبط با بازخورد ارائه می دهد. این دیدگاه برای تجزیه و تحلیل و گزارش در مورد رضایت مشتری و خدمات خاصی که بازخورد دریافت کرده اند مفید است.

```
--CustomerFeedbackAndServicesView view
CREATE VIEW CustomerFeedbackAndServicesView AS
SELECT
    C.Name AS CustomerName,
    F.Rating,
    F.Comments,
    S.ServiceName
FROM Customers C
JOIN Feedbacks F ON C.CustomerID = F.CustomerID
JOIN Services S ON F.ServiceID = S.ServiceID;
GO
```

٤. ServiceSummaryView View:

عملیات: این ویو خروجی های تابع GetServiceSummary که بالاتر نحوه کار آن توضیح داده شده است را برمیگرداند تا از نتایج آن استفاده شود در صورت نیاز.

```
--ServiceSummaryView view
CREATE VIEW ServiceSummaryView AS
SELECT
    ServiceID,
    ServiceName,
    NumberOfReviews,
    AverageServiceRating,
    TotalSales
FROM GetServiceSummary();
GO
```

٥. EmployeesView View:

عملیات: این ویو ستون های مشخص و مورد نیاز از جدول کارمندان را برمیگرداند تا اطلاعات جامعی از کارمندان موجود بدست آورد.

```
--EmployeesView view
CREATE VIEW EmployeesView AS
SELECT
    E.EmployeeID,
    E.Name AS EmployeeName,
    E.Phone AS EmployeePhone,
    E.Email AS EmployeeEmail,
    E.Role AS EmployeeRole
FROM Employees E;
GO
```

٦. DatabaseInfo View:

عملیات: این ویو اطلاعاتی در مورد آبجکت های مختلف پایگاه داده، به ویژه جداول، توابع، پراسیجرها، تریگرها و ویوها ارائه می دهد .این شامل جزئیاتی مانند نام آبجکت و نوع آن و برای جداول، تعداد ردیف است. این ویو یک نمای کلی در سطح بالا از اطلاعات ساختاری پایگاه داده و برخی از آمارهای اولیه در مورد تعداد ردیف جداول ارائه می دهد.

```sql
CREATE VIEW DatabaseInfo AS
WITH ObjectDetails AS (
    SELECT
        schema_name(schema_id) AS SchemaName,
        name AS ObjectName,
        CASE
            WHEN type = 'U' THEN 'Table'
            WHEN type = 'FN' THEN 'Function'
            WHEN type = 'P' THEN 'Procedure'
            WHEN type = 'TR' THEN 'Trigger'
            WHEN type = 'V' THEN 'View'
            ELSE 'Unknown'
        END AS ObjectType
    FROM
        sys.objects
    WHERE
        type IN ('U', 'FN', 'P', 'TR', 'V')
)
SELECT
    od.ObjectName,
    od.ObjectType,
    CASE
        WHEN od.ObjectType = 'Table' THEN
            (SELECT SUM(rows) FROM sys.partitions WHERE object_id = OBJECT_ID(od.SchemaName + '.' + od.ObjectName))
        ELSE
            NULL
    END AS [RowCount]
FROM
    ObjectDetails od;
GO
```