Examentraining 1

Realiseren

hoofdstuk

1

Stored procedures en views

Algemene informatie

Onderwerp	Stored procedures en views (SQL Server)
Leerdoel(en)	1. De student is bekend met stored procedures binnen SQL Server 2. De student is bekend met views binnen SQL Server
Vereiste voorkennis	1. De student heeft uitgebreide OOP voorkennis. 2. De student heeft uitgebreide MVC architectuur voorkennis
Kwalificatiedossier	☐ B1-K1-W1: Plant werkzaamheden en bewaakt de voortgang ☐ B1-K1-W2: Ontwerpt software ☒ B1-K1-W3: Realiseert (onderdelen van) software ☐ B1-K1-W4: Test software ☐ B1-K1-W5: Doet verbetervoorstellen voor de software ☐ B1-K2-W1: Voert overleg ☐ B1-K2-W2: Presenteert het opgeleverde werk ☐ B1-K2-W3: Reflecteert op het werk

Inhoudsopgave

Introductie

Je hebt in alle voorgaande realiseren lessen geleerd hoe je door middel van query's gegevens uit tabellen kan lezen, aanpassen, verwijderen en toevoegen. In dit hoofdstuk gaan we kijken naar alternatieven hiervoor. Je leert wat een Stored Procedure is en wat een View is en hoe je dit kunt gebruiken bij het bouwen van een applicatie.

Inhoud

1. Stored procedure

Een stored procedure is feitelijk een stukje code wat je in de database opslaat. Deze code is natuurlijk niet geschreven in C# maar in SQL (de taal voor communicatie met databases). Je kan een stored procedure dus vergelijken met een methode in C#.

Een methode in C# om een product op te slaan zou er als volgt uit kunnen zien:

```
public int Create(ProductModel product)
{
    int affectedRows;
    using (SqlConnection con = new SqlConnection(connectionString))
    {
        string sqlQuery = " INSERT INTO Product " +
                          "VALUES(@ProductName, @SupplierID, @UnitPrice, NULL); ";
        using (SqlCommand command = new SqlCommand(sqlQuery, con))
        {
            command.Parameters.AddWithValue("ProductName", product.ProductName);
            command.Parameters.AddWithValue("UnitPrice", product.UnitPrice);
            command.Parameters.AddWithValue("SupplierID", product.Supplier.Id);
            con.Open();
            affectedRows = command.ExecuteNonQuery();
        }
    }
    return affectedRows;
}
```

Dit zou je als volgt ook door een Stored procedure kunnen laten uitvoeren. Let op! Het verschil is dat deze Stored procedure niet als resultaat heeft hoeveel rijen er zijn aangepast, maar hij geeft het Id van het aangemaakte product als resultaat!

```
CREATE PROCEDURE CreateProduct
    @ProductName    NVARCHAR(255),
    @SupplierID     INT,
    @UnitPrice      DECIMAL(18, 2)
AS
BEGIN
    INSERT INTO Product (ProductName, SupplierID, UnitPrice)
    VALUES (@ProductName, @SupplierID, @UnitPrice);

    DECLARE @ProductId INT
    SET @ProductId =(SELECT SCOPE_IDENTITY() AS NewProductId)
    RETURN @ProductId; -- Dit geeft het ID terug van het ingevoegde product.
END
```

1.1 Voordelen stored procedures

Hier zijn enkele situaties waarin je stored procedures kunt overwegen:

Gegevensbeheer en beveiliging: Stored procedures kunnen helpen bij het beheren van complexe databasebewerkingen en het afdwingen van beveiligingsbeleid. Ze stellen je in staat om de toegang tot de database opgeslagen procedures te beheren en ervoor te zorgen dat alleen geautoriseerde gebruikers de bewerkingen kunnen uitvoeren.

Optimalisatie van databasequery's: Stored procedures kunnen worden geoptimaliseerd en vooraf worden gecompileerd, wat de queryprestaties kan verbeteren, vooral bij veelgebruikte en complexe bewerkingen. Dit kan de algehele prestaties van je applicatie verbeteren.

Afscherming van gegevenslaag: Door stored procedures te gebruiken, kun je de complexiteit van de database-interacties in de gegevenslaag van je applicatie verbergen. Dit verbetert de modulariteit van je code en maakt het gemakkelijker om wijzigingen in de databasestructuur aan te brengen zonder de applicatielogica te beïnvloeden.

Transactiebeheer: Je kunt stored procedures gebruiken om transacties in je database te beheren. Dit is handig als je meerdere databasebewerkingen in één transactie wilt uitvoeren en ervoor wilt zorgen dat ze allemaal slagen of mislukken als één geheel.

1.2 Aanroepen Stored Prodedure vanuit C#

Als je bovenstaande Stored Procedure vanuit C# code zou willen aanroepen kan dat op de volgende manier:

```
public int CreateProduct(ProductModel product)
{
    int newProductID = 0; // Hier houden we het ID van het nieuw ingevoegde product bij.

    using (SqlConnection con = new SqlConnection(connectionString))
    {
        using (SqlCommand command = new SqlCommand("CreateProduct", con))
        {
             // Geef aan dat we een stored procedure uitvoeren.
            command.CommandType = CommandType.StoredProcedure;

            // Voeg de parameters toe aan de stored procedure.
            command.Parameters.AddWithValue("@ProductName", product.ProductName);
            command.Parameters.AddWithValue("@SupplierID", product.Supplier.Id);
            command.Parameters.AddWithValue("@UnitPrice", product.UnitPrice);

            // Voeg een parameter toe om het nieuw ingevoegde ID op te vangen.
            SqlParameter outputParameter = new SqlParameter("@NewProductID", SqlDbType.Int);
            outputParameter.Direction = ParameterDirection.ReturnValue;

             command.Parameters.Add(outputParameter);

            con.Open();
            command.ExecuteNonQuery();

            // Haal het nieuw ingevoegde ID op uit de outputparameter.
            newProductID = (int)outputParameter.Value;
        }
    }

    return newProductID; // Retourneer het ID van het nieuw ingevoegde product.
}
```

Je ziet dat de code heel erg lijkt op de code die je gebruikt als je een losse query zou uitvoeren. Er is wel wat verschil, zo maak je in bovenstaande code voor het eerst gebruik van een output parameter, ofwel een parameter die gevuld wordt door de stored procedure!

 Je kunt nu **oefening 1.1** maken.

2. Views

In SQL Server is een "view" een virtuele tabel die is gebaseerd op een query (een SELECT-instructie) over een of meer tabellen. Het belangrijkste doel van een view is om een gemakkelijke en veilige manier te bieden om gegevens uit een of meer tabellen op te halen, zonder dat je de onderliggende tabelstructuur hoeft te kennen. Hier zijn enkele belangrijke punten over views:

1. Virtuele Tabel: Een view is geen fysieke tabel in de database; het is een virtuele tabel die wordt gegenereerd op basis van een query.

2. Beperking van Gegevens: Met een view kun je een beperkt deel van de gegevens in een tabel tonen. Je kunt bepaalde kolommen selecteren en rijen filteren op basis van bepaalde voorwaarden.

3. Veiligheid en Autorisatie: Views kunnen worden gebruikt om de toegang tot gevoelige gegevens te beperken. Je kunt bepaalde gebruikers toegang geven tot een view zonder hen toegang te geven tot de onderliggende tabellen.

4. Vereenvoudiging van Query's: Views kunnen complexe query's vereenvoudigen. Als je vaak dezelfde complexe query moet uitvoeren, kun je deze in een view opslaan en de view telkens opnieuw gebruiken.

5. Logica Opsplitsen: Views kunnen worden gebruikt om bedrijfslogica op te splitsen in kleinere, beheersbare delen. Dit verbetert de leesbaarheid en onderhoudbaarheid van je SQL-code.

6. Aggregaties en Berekeningen: Je kunt views gebruiken om aggregaties (bijvoorbeeld totalen of gemiddelden) en berekeningen uit te voeren op gegevens in tabellen.

7. Gegevenscentralisatie: Met views kun je gegevens van meerdere tabellen centraliseren en presenteren als één virtuele tabel, waardoor complexe rapporten en analyses mogelijk worden.

2.1 Eenvoudig voorbeeld van een view in SQL

Stel dat je een database hebt met een "Orders"-tabel en een "Customers"-tabel. Je wilt een view maken die de namen van klanten en bijbehorende bestellingen toont:

```
CREATE VIEW CustomerOrders AS
SELECT Customers.CustomerName, Orders.OrderID, Orders.OrderDate
FROM Customers
INNER JOIN Orders ON Customers.CustomerID = Orders.CustomerID;
```

Na het maken van deze view kun je eenvoudig de namen van klanten en bijbehorende bestellingen ophalen zonder de onderliggende tabelstructuren te kennen:

```
SELECT * FROM CustomerOrders;
```

Dit is een eenvoudig voorbeeld van hoe views kunnen worden gebruikt om gegevens op een meer georganiseerde en veilige manier te benaderen in SQL Server.

2.2 Tweede voorbeeld van een view in SQL

Laten we aannemen dat de "Product"-tabel een kolom "SupplierID" bevat die verwijst naar de "Supplier"-tabel op basis van het "SupplierID" veld. Hier is een voorbeeld van zo'n view:

```
CREATE VIEW ProductSupplierInfo AS
SELECT
    P.ProductID,
    P.ProductName,
    P.UnitPrice,
    S.SupplierName,
    S.SupplierAddress
FROM
    Product AS P INNER JOIN Supplier AS S
ON
    P.SupplierID = S.SupplierID;
```

In dit voorbeeld wordt de view "ProductSupplierInfo" gemaakt. Deze view haalt informatie op uit zowel de "Product"-tabel als de "Supplier"-tabel. Het koppelt producten aan hun respectievelijke leveranciers op basis van de "SupplierID" en toont de volgende gegevens:

ProductID: Het unieke identificatienummer van het product.
ProductName: De naam van het product.
UnitPrice: De prijs van het product.
SupplierName: De naam van de leverancier.
SupplierAddress: Het adres van de leverancier.

Met deze view kun je eenvoudig gegevens ophalen die informatie over producten en hun leveranciers combineren, zonder dat je complexe JOIN-operaties in elke query hoeft uit te voeren.

Zo'n select uit een View levert een simpelere, overzichtelijkere query op in je code. Je krijgt dan in de Read method deze regel code:

```
string query = "SELECT * FROM ProductSupplierInfo";
using (SqlCommand command = new SqlCommand(query, con));
```

in plaats van deze code:

```
string query = "SELECT P.ProductID, " +
    "P.ProductName," +
    "P.UnitPrice," +
    "S.SupplierName," +
    "S.SupplierAddress " +
    "FROM Product AS P " +
    "   INNER JOIN Supplier AS S " +
    "ON P.SupplierID = S.SupplierID;";
using (SqlCommand command = new SqlCommand(query, con));
```

 Je kunt nu **oefening 1.2** maken.