```
/*
    a) Создайте таблицу Production.ProductModelHst,
       которая будет хранить информацию об изменениях в таблице Production.ProductModel.
       Обязательные поля, которые должны присутствовать в таблице:
           ID – первичный ключ IDENTITY(1,1);
           Action – совершенное действие (insert, update или delete);
           ModifiedDate – дата и время, когда была совершена операция;
           SourceID – первичный ключ исходной таблицы;
           UserName – имя пользователя, совершившего операцию.
       Создайте другие поля, если считаете их нужными.
*/
CREATE TABLE Production.ProductModelHst
    (
        ID INT IDENTITY(1, 1) PRIMARY KEY,
        Action VARCHAR(6) CHECK (ACTION IN ('INSERT', 'UPDATE', 'DELETE')),
        ModifiedDate DATETIME,
        SourceID INT,
        UserName VARCHAR(20)
    );
```

Рисунок 1.1 – Задание 1-а (Решение)

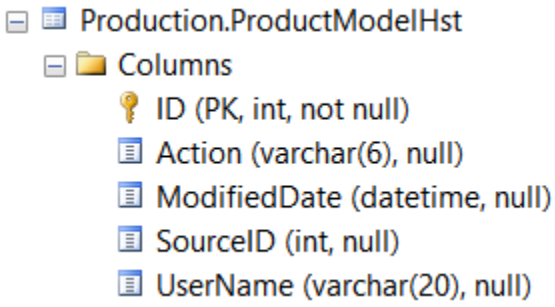

Рисунок 1.2 – Задание 1-а (Результат)

```
/*
    b) Создайте один AFTER триггер для трех операций INSERT, UPDATE, DELETE для таблицы Production.ProductModel.
       Триггер должен заполнять таблицу Production.ProductModelHst с указанием типа операции в поле Action
       в зависимости от оператора, вызвавшего триггер.
*/
CREATE TRIGGER ProductModel_Insert
    ON Production.ProductModel
    AFTER INSERT
AS
    INSERT INTO Production.ProductModelHst
    SELECT
        'INSERT',
        GETDATE(),
        INSERTED.ProductModelID,
        CURRENT_USER
    FROM INSERTED;
GO

CREATE TRIGGER ProductModel_Update
    ON Production.ProductModel
    AFTER UPDATE
AS
    INSERT INTO Production.ProductModelHst
    SELECT
        'UPDATE',
        GETDATE(),
        DELETED.ProductModelID,
        CURRENT_USER
    FROM DELETED;
GO

CREATE TRIGGER ProductModel_Delete
    ON Production.ProductModel
    AFTER DELETE
AS
    INSERT INTO Production.ProductModelHst
    SELECT
        'DELETE',
        GETDATE(),
        DELETED.ProductModelID,
        CURRENT_USER
    FROM DELETED;
GO
```

Рисунок 1.3 – Задание 1-b (Решение)

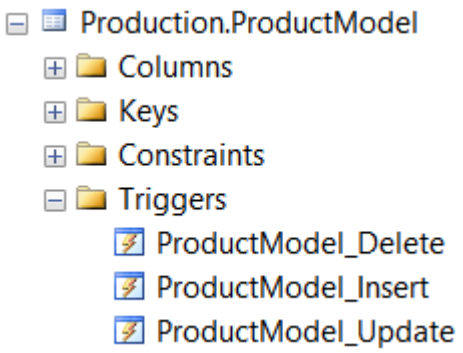

Рисунок 1.4 – Задание 1-b (Результат)

```sql
/*
    c) Создайте представление VIEW, отображающее все поля таблицы Production.ProductModel.
*/
CREATE VIEW Production.ProductModelView
AS
    SELECT
        *
    FROM Production.ProductModel;
GO
```

Рисунок 1.5 – Задание 1-c (Решение)

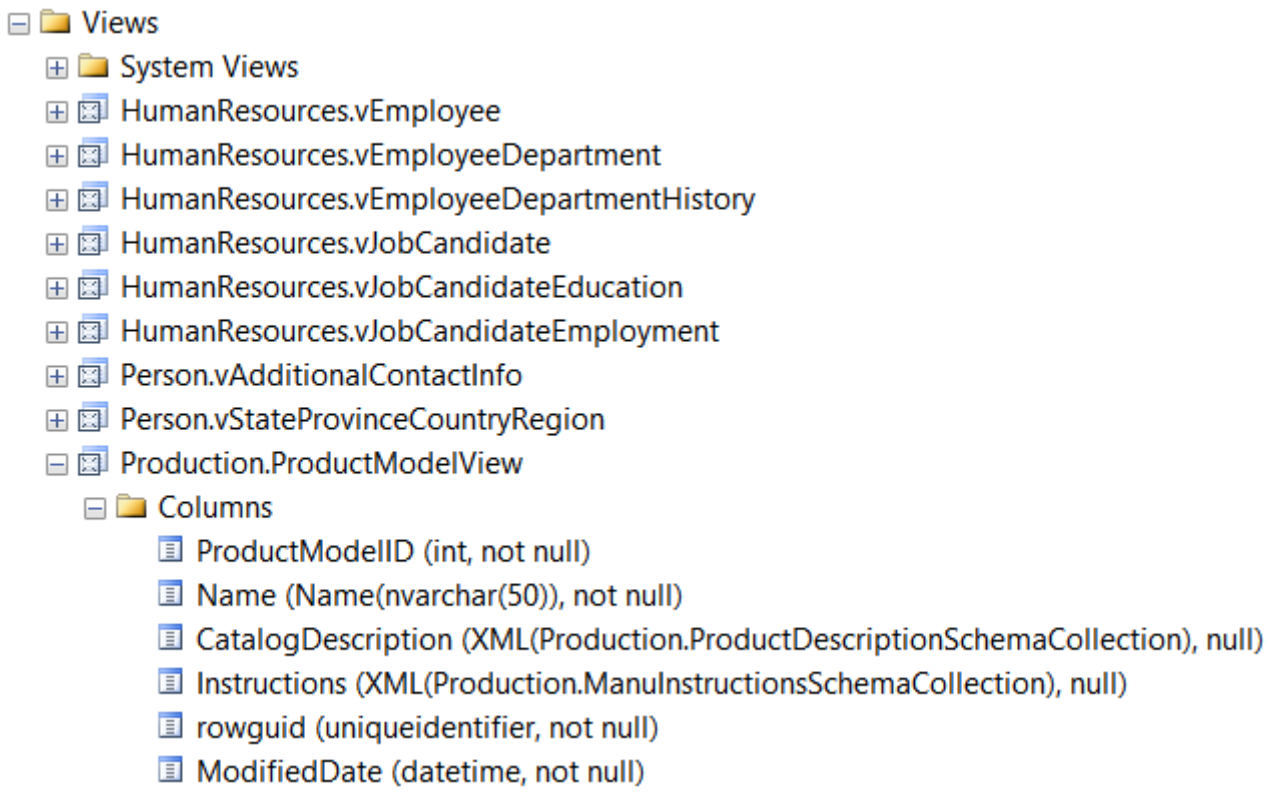

Рисунок 1.6 – Задание 1-c (Результат)

```sql
/*
    d) Вставьте новую строку в Production.ProductModel через представление.
       Обновите вставленную строку.
       Удалите вставленную строку.
       Убедитесь, что все три операции отображены в Production.ProductModelHst.
*/
INSERT INTO Production.ProductModelView
    (
        Name
    )
VALUES
    (
        'Test product'
    );

UPDATE Production.ProductModelView
SET Name = 'Updated name'
WHERE Name = 'Test product';

DELETE FROM Production.ProductModelView
WHERE Name = 'Updated name';

SELECT *
FROM Production.ProductModelHst;
```

Рисунок 1.7 – Задание 1-d (Решение)

	ID	Action	ModifiedDate	Source...	UserNa...
1	29	INSERT	2020-10-31 22:31:37.770	152	dbo
2	30	UPDATE	2020-10-31 22:31:40.877	152	dbo
3	31	DELETE	2020-10-31 22:31:43.197	152	dbo

Рисунок 1.8 – Задание 1-d (Результат)

```sql
/*
    a) Создайте представление VIEW, отображающее данные из таблиц Production.ProductModel,
       Production.ProductModelProductDescriptionCulture, Production.Culture и Production.ProductDescription.
       Сделайте невозможным просмотр исходного кода представления.
       Создайте уникальный кластерный индекс в представлении по полям ProductModelID,CultureID.
*/
CREATE VIEW Production.ProductModelProductDescriptionCultureView
    WITH ENCRYPTION, SCHEMABINDING
AS
    SELECT
        Model.ProductModelID,
        Description.ProductDescriptionID,
        Culture.CultureID,
        UnionTable.ProductModelID AS UnionProductModelID,
        UnionTable.ProductDescriptionID AS UnionDescriptionID,
        UnionTable.CultureID AS UnionCultureID,
        UnionTable.ModifiedDate AS ProductModelProductDescriptionCultureModifiedDate,
        Model.Name AS ProductModelName,
        Model.CatalogDescription,
        Model.Instructions,
        Model.rowguid AS ProductModelRowguid,
        Model.ModifiedDate AS ProductModelModifiedDate,
        Culture.Name AS CultureName,
        Culture.ModifiedDate AS CultureModifiedDate,
        Description.Description,
        Description.rowguid AS ProductDescriptionRowguid,
        Description.ModifiedDate AS ProductDescriptionModifiedDate
    FROM Production.ProductModelProductDescriptionCulture AS UnionTable
        JOIN Production.ProductModel AS Model
        ON UnionTable.ProductModelID = Model.ProductModelID
        JOIN Production.Culture AS Culture
        ON UnionTable.CultureID = Culture.CultureID
        JOIN Production.ProductDescription AS Description
        ON UnionTable.ProductDescriptionID = Description.ProductDescriptionID;
GO

CREATE UNIQUE CLUSTERED INDEX I_ProductModelID_CultureID
ON Production.ProductModelProductDescriptionCultureView
    (
        ProductModelID,
        CultureID
    );
GO
```

Рисунок 2.1 – Задание 2-а (Решение)

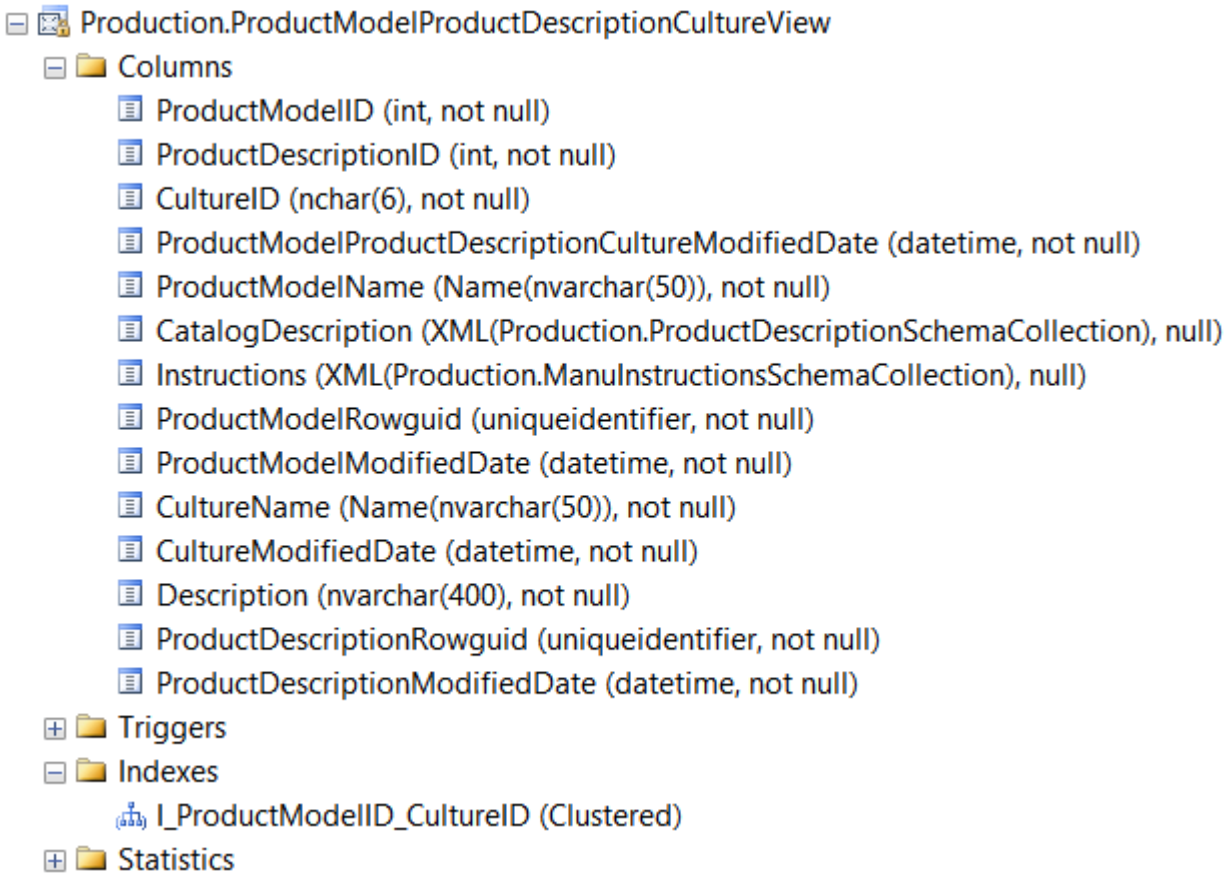

Рисунок 2.2 – Задание 2-a (Результат)

```
/*
    b) Создайте три INSTEAD OF триггера для представления на операции INSERT, UPDATE, DELETE.
       Каждый триггер должен выполнять соответствующие операции в таблицах Production.ProductModel,
       Production.ProductModelProductDescriptionCulture, Production.Culture и Production.ProductDescription.
       Обновление не должно происходить в таблице Production.ProductModelProductDescriptionCulture.
       Удаление строк из таблиц Production.ProductModel, Production.Culture и Production.ProductDescription производите только в том случае,
       если удаляемые строки больше не ссылаются на Production.ProductModelProductDescriptionCulture.
*/
```

Рисунок 2.3 – Задание 2-b (Задание)

```
CREATE TRIGGER Instead_of_Insert
    ON Production.ProductModelProductDescriptionCultureView
    INSTEAD OF INSERT
AS
    DECLARE @ProductModelIDTable
    TABLE
        (
            ID INT IDENTITY(1,1),
            ProductModelID INT
        )

    INSERT INTO Production.ProductModel
        (
            Name,
            CatalogDescription,
            Instructions
        )
    OUTPUT INSERTED.ProductModelID
        INTO @ProductModelIDTable (ProductModelID)
    SELECT
        INSERTED.ProductModelName,
        INSERTED.CatalogDescription,
        INSERTED.Instructions
    FROM INSERTED

    DECLARE @CultureIDTable
    TABLE
        (
            ID INT IDENTITY(1,1),
            CultureID NCHAR(6)
        )

    INSERT INTO Production.Culture
        (
            CultureID,
            Name
        )
    OUTPUT INSERTED.CultureID
        INTO @CultureIDTable (CultureID)
    SELECT
        INSERTED.CultureID,
        INSERTED.CultureName
    FROM INSERTED
```

Рисунок 2.4 – Задание 2-b – Instead of Insert Trigger (Начало)

```
    DECLARE @ProductDescriptionIDTable
    TABLE
        (
            ID INT IDENTITY(1,1),
            ProductDescriptionID INT
        )

    INSERT INTO Production.ProductDescription
        (
            Description
        )
    OUTPUT INSERTED.ProductDescriptionID
        INTO @ProductDescriptionIDTable (ProductDescriptionID)
    SELECT
        INSERTED.Description
    FROM INSERTED

    INSERT INTO Production.ProductModelProductDescriptionCulture
        (
            ProductModelID,
            ProductDescriptionID,
            CultureID
        )
    SELECT
        ModelID.ProductModelID,
        DescriptionID.ProductDescriptionID,
        CultureID.CultureID
    FROM @ProductModelIDTable AS ModelID
        JOIN @CultureIDTable AS CultureID
        ON ModelID.ID = CultureID.ID
        JOIN @ProductDescriptionIDTable AS DescriptionID
        ON ModelID.ID = DescriptionID.ID
GO
```

Рисунок 2.5 – Задание 2-b – Instead of Insert Trigger (Конец)

```
CREATE TRIGGER Instead_of_Update
    ON Production.ProductModelProductDescriptionCultureView
    INSTEAD OF UPDATE
AS
    UPDATE Production.ProductModel
    SET
        Name = (
            SELECT ProductModelName
            FROM INSERTED
            WHERE Production.ProductModel.ProductModelID = INSERTED.ProductModelID
        ),
        CatalogDescription = (
            SELECT CatalogDescription
            FROM INSERTED
            WHERE Production.ProductModel.ProductModelID = INSERTED.ProductModelID
        ),
        Instructions = (
            SELECT Instructions
            FROM INSERTED
            WHERE Production.ProductModel.ProductModelID = INSERTED.ProductModelID
        )
    WHERE Production.ProductModel.ProductModelID IN
        (
            SELECT
                ProductModelID
            FROM INSERTED
        )
```

Рисунок 2.6 – Задание 2-b – Instead of Update Trigger (Начало)

```
    UPDATE Production.Culture
    SET
        CultureID = (
            SELECT CultureID
            FROM INSERTED
            WHERE Production.Culture.CultureID = INSERTED.CultureID
        ),
        Name = (
            SELECT CultureName
            FROM INSERTED
            WHERE Production.Culture.CultureID = INSERTED.CultureID
        )
    WHERE Production.Culture.CultureID IN
        (
            SELECT
                CultureID
            FROM INSERTED
        )

    UPDATE Production.ProductDescription
    SET
        Description = (
            SELECT Description
            FROM INSERTED
            WHERE Production.ProductDescription.ProductDescriptionID = INSERTED.ProductDescriptionID
        )
    WHERE Production.ProductDescription.ProductDescriptionID IN
        (
            SELECT
                ProductDescriptionID
            FROM INSERTED
        )
GO
```

Рисунок 2.7 – Задание 2-b – Instead of Update Trigger (Конец)

```
CREATE TRIGGER Instead_of_Delete
    ON Production.ProductModelProductDescriptionCultureView
    INSTEAD OF DELETE
AS
    IF
        (
            NOT EXISTS
                (
                    SELECT ProductModelID
                    FROM Production.ProductModelProductDescriptionCulture AS UnionTable
                    WHERE UnionTable.ProductModelID IN (SELECT ProductModelID FROM DELETED)
                ) AND
            NOT EXISTS
                (
                    SELECT CultureID
                    FROM Production.ProductModelProductDescriptionCulture AS UnionTable
                    WHERE UnionTable.CultureID IN (SELECT CultureID FROM DELETED)
                ) AND
            NOT EXISTS
                (
                    SELECT ProductDescriptionID
                    FROM ProductModelProductDescriptionCulture AS UnionTable
                    WHERE UnionTable.ProductDescriptionID IN (SELECT ProductDescriptionID FROM DELETED)
                )
        )
```

Рисунок 2.8 – Задание 2-b – Instead of Delete Trigger (Начало)

```
            BEGIN
                DELETE FROM Production.ProductModel
                WHERE ProductModelID IN
                    (
                        SELECT ProductModelID
                        FROM DELETED
                    )

                DELETE FROM Production.Culture
                WHERE CultureID IN
                    (
                        SELECT CultureID
                        FROM DELETED
                    )

                DELETE FROM Production.ProductDescription
                WHERE ProductDescriptionID IN
                    (
                        SELECT ProductDescriptionID
                        FROM DELETED
                    )
            END
        GO
```

Рисунок 2.9 – Задание 2-b – Instead of Delete Trigger (Конец)

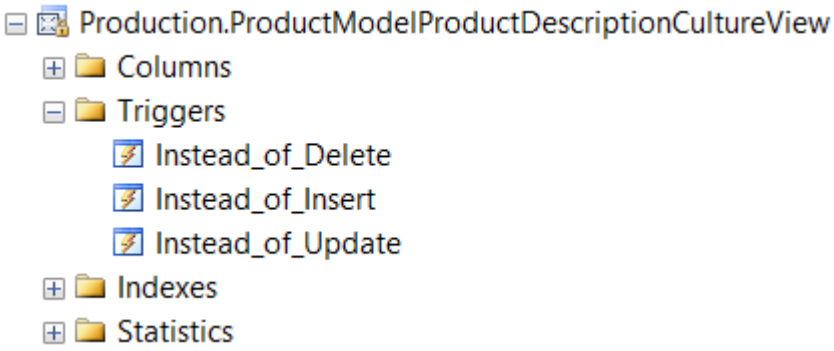

Рисунок 2.10 – Задание 2-b (Результат)

```
/*
    c) Вставьте новую строку в представление, указав новые данные для ProductModel, Culture и ProductDescription.
       Триггер должен добавить новые строки в таблицы Production.ProductModel,
       Production.ProductModelProductDescriptionCulture, Production.Culture и Production.ProductDescription.
       Обновите вставленные строки через представление.
       Удалите строки.
*/
INSERT INTO Production.ProductModelProductDescriptionCultureView
    (
        CultureID,
        ProductModelName,
        CultureName,
        Description
    )
VALUES
    (
        'TESTID',
        'TEST PRODUCT NAME',
        'TEST CULTURE NAME',
        'DESCRIPTION'
    ),
    (
        'TID',
        'TEST 2',
        'TEST 2',
        'TEST 2'
    );
GO

UPDATE Production.ProductModelProductDescriptionCultureView
SET
    Production.ProductModelProductDescriptionCultureView.Description = 'NEW MODEL Description'
WHERE
    Production.ProductModelProductDescriptionCultureView.ProductModelName = 'TEST PRODUCT NAME' OR
    Production.ProductModelProductDescriptionCultureView.ProductModelName = 'TEST 2';
GO

DELETE FROM Production.ProductModelProductDescriptionCulture
WHERE CultureID = 'TESTID' OR CultureID = 'TID'

DELETE FROM Production.ProductModelProductDescriptionCultureView
WHERE Description = 'NEW MODEL Description';
GO
```

Рисунок 2.11 – Задание 2-c (Решение)

	ProductMode...	ProductDescriptio...	Culture...	ModifiedDate	ProductMode...	Name	CatalogDescription	Instructio...	rowguid
756	126	2004	zh-cht	2007-06-01 00:00:00.000	126	LL Road Rear W...	NULL	NULL	95946B
757	127	2005	en	2007-06-01 00:00:00.000	127	Rear Derailleur	NULL	NULL	F9327E
758	127	2006	ar	2007-06-01 00:00:00.000	127	Rear Derailleur	NULL	NULL	F9327E
759	127	2007	fr	2007-06-01 00:00:00.000	127	Rear Derailleur	NULL	NULL	F9327E
760	127	2008	th	2007-06-01 00:00:00.000	127	Rear Derailleur	NULL	NULL	F9327E
761	127	2009	he	2007-06-01 00:00:00.000	127	Rear Derailleur	NULL	NULL	F9327E
762	127	2010	zh-cht	2007-06-01 00:00:00.000	127	Rear Derailleur	NULL	NULL	F9327E
763	150	2019	TESTID	2020-10-31 20:52:59.160	150	TEST PRODUCT...	NULL	NULL	8252C7
764	151	2020	TID	2020-10-31 20:52:59.160	151	TEST 2	NULL	NULL	001004

Рисунок 2.12 – Задание 2-с (Результат вставки)

	ProductMode...	ProductDescriptio...	Culture...	ModifiedDate	Name	ModifiedDate	Description
755	126	2003	he	2007-06-01 00:00:00.000	LL Road Rear W...	2006-06-01 00:00:00.000	גלגל חלופי קדמי לרכיבת כביש לרוכב המתחיל.
756	126	2004	zh-cht	2007-06-01 00:00:00.000	LL Road Rear W...	2006-06-01 00:00:00.000	适用于入门级骑乘者的公路型备用前轮。
757	127	2005	en	2007-06-01 00:00:00.000	Rear Derailleur	2007-06-01 00:00:00.000	Wide-link design.
758	127	2006	ar	2007-06-01 00:00:00.000	Rear Derailleur	2007-06-01 00:00:00.000	تصميم عريض الوصلات.
759	127	2007	fr	2007-06-01 00:00:00.000	Rear Derailleur	2007-06-01 00:00:00.000	Conception liaison large.
760	127	2008	th	2007-06-01 00:00:00.000	Rear Derailleur	2007-06-01 00:00:00.000	การออกแบบให้มีจุดเชื่อมกว้าง
761	127	2009	he	2007-06-01 00:00:00.000	Rear Derailleur	2007-06-01 00:00:00.000	עיצוב רחב-חוליות.
762	127	2010	zh-cht	2007-06-01 00:00:00.000	Rear Derailleur	2007-06-01 00:00:00.000	宽连杆设计。
763	150	2019	TESTID	2020-10-31 20:52:59.160	TEST PRODUCT...	2020-10-31 20:52:59.157	NEW MODEL Description
764	151	2020	TID	2020-10-31 20:52:59.160	TEST 2	2020-10-31 20:52:59.157	NEW MODEL Description

Рисунок 2.13 – Задание 2-с (Результат обновление)

	ProductMode...	ProductDescriptio...	Culture...	ModifiedDate	ProductMode...	Name	CatalogDescription	Instructio...	row
754	126	2002	th	2007-06-01 00:00:00.000	126	LL Road Rear W...	NULL	NULL	959
755	126	2003	he	2007-06-01 00:00:00.000	126	LL Road Rear W...	NULL	NULL	959
756	126	2004	zh-cht	2007-06-01 00:00:00.000	126	LL Road Rear W...	NULL	NULL	959
757	127	2005	en	2007-06-01 00:00:00.000	127	Rear Derailleur	NULL	NULL	F93
758	127	2006	ar	2007-06-01 00:00:00.000	127	Rear Derailleur	NULL	NULL	F93
759	127	2007	fr	2007-06-01 00:00:00.000	127	Rear Derailleur	NULL	NULL	F93
760	127	2008	th	2007-06-01 00:00:00.000	127	Rear Derailleur	NULL	NULL	F93
761	127	2009	he	2007-06-01 00:00:00.000	127	Rear Derailleur	NULL	NULL	F93
762	127	2010	zh-cht	2007-06-01 00:00:00.000	127	Rear Derailleur	NULL	NULL	F93

Рисунок 2.14 – Задание 2-с (Результат удаления)