**Федеральное государственное автономное образовательное учреждение  
высшего профессионального образования**

**НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ТЕХНОЛОГИЧЕСКИЙ  
УНИВЕРСИТЕТ «МИСИС»**

Отчет по теме:  
«База данных автосалона»

по курсу: «Базы Данных»

Выполнила:

Игнатова Мария Евгеньевна

Группа: БИВТ-23-1

Проверила: Валова А.А.

Москва, 2024

**Оглавление**

[**1.Постановка задачи** 3](#_Toc185559001)

[**2. Описание структуры базы данных** 3](#_Toc185559002)

[**2.1 Вербальная модель** 3](#_Toc185559003)

[**2.2 Реляционная модель** 4](#_Toc185559004)

[**2.3 Анализ функциональных зависимостей** 7](#_Toc185559005)

[**3. Заполнение базы данных информацией** 8](#_Toc185559006)

[**4. Описание представлений** 11](#_Toc185559007)

[**5.** **Описание функций** 19](#_Toc185559008)

[**6.** **Описание хранимых процедур** 22](#_Toc185559009)

[**7.** **Описание триггеров** 34](#_Toc185559010)

[**8. Примеры работы с БД с использованием созданных объектов** 40](#_Toc185559011)

[**9.** **Список литературы** 45](#_Toc185559012)

# **1.Постановка задачи**

**Номер варианта: 27.**

**Предметная область: Автосалон.**

Описание:

База данных предназначена для управления информацией о автосалоне, включая данные о клиентах, менеджерах, сделках, продаваемых машинах, автозапчастях, поставщиках и производителях машин и автозапчастей, филиалах сбыта.

# **2. Описание структуры базы данных**

## **2.1 Вербальная модель**

Реализованная база данных содержит 11 сущностей.  
**1) Информация о машинах**

Здесь содержится информация о машине, продаваемой в автосалоне: модель машины, идентификатор комплектации, цена за базовую комплектацию, идентификатор поставщика машины, идентификатор автосалона, в котором машина находится, производитель машины, статус продажи: куплена машина или доступна к покупке.

**2) Комплектации машин**

Здесь содержится информация о всех комплектациях машин: идентификатор комплектации, название, вес, цвет машины, тип трансмиссии, тип кузова, тип двигателя, повышающий цену коэффициент.

**3) Информация о производителях**

Здесь содержится информация о производителе машин: идентификатор производителя, полное название, адрес, страна производителя, официальный сайт, телефон.

**4) Информация о филиалах**

Здесь содержится информация о филиалах компании: идентификатор филиала, полное название, адрес, номер телефон.

**5) Информация о поставщиках**

Здесь содержится информация о поставщиках, предоставляющих машины и запчасти: идентификатор поставщика, полное название, адрес, номер телефона, адрес электронной почты.

**6) Информация о автозапчастях**

Здесь содержится информация о автозапчастях: идентификатор запчасти, название категории, к которой относится запчасть, название запчасти, количество на складе, идентификатор поставщика, модель машины, к которой эта запчасть подходит.

**7) Информация о заказе**

Здесь содержится информация о номере заказа, дате отправки и дате доставки, методе доставки и общей сумме заказа.

**8) Информация о деталях заказа**

Здесь содержится информация о деталях заказа: идентификатор номера заказа, идентификатор детали заказа, связанный с конкретной заказываемой автозапчастью, идентификатор запчасти, цена за запчасть, количество, сумма заказа по одному виду продукции.

**9) Информация о менеджерах**

Здесь содержится информация о менеджерах: идентификатор менеджера, полное ФИО, филиал, в котором работает сотрудник, количество заключенных сделок.

**10) Информация о клиентах**

Здесь содержится информация о клиентах автосалона: идентификатор клиента, полное ФИО, адрес, номер телефона, адрес электронной почты, паспортные данные.

**11) Информация о продажах**  
Здесь содержится информация о продажах: идентификатор продажи, идентификатор машины, идентификатор менеджера, идентификатор клиента, дата продажи, сумма продажи, статус оплаты, дата отправки и дата доставки машины.

Все таблицы содержат первичные ключи для уникальной идентификации записей, а также внешние ключи для обеспечения связности данных между таблицами.

## **2.2 Реляционная модель**

Соответствие отношений к сущностям:

**1) Cars (Информация о машинах)**

Foreign key: CompleteSetID ( идентификатор комплектации) из отношения CompleteSets,

SupplierD ( идентификатор поставщика) из отношения Suppliers, CarDealershipID (идентификатор филиала) из отношения CarDealerships, ManufacturerID (идентификатор производителя) из отношения Manufacturers.

Primary key: CarID (идентификатор машины)

**2) CompleteSets ( Информация о комплектациях машин)**

Primary key: CompleteSetID (идентификатор комплектации)

**3) Manufacturers (информация о производителях)**

Primary key: ManufacturerID (идентификатор производителя)

**4) CarDealerships (информация о филиалах)**

Primary key: CarDealershipID (идентификатор производителя)

**5) Suppliers (информация о поставщиках)**

Primary key: SupplierID (идентификатор поставщика)

**6) AutoParts (информация о автозапчастях)**

Foreign key: SupplierID ( идентификатор поставщика) из отношения Suppliers

Primary key: AutoPartID (идентификатор автозапчасти)

**7) AutoPartOrderHeader (информация о заказе)**

Primary key: OrderID (идентификатор заказа)

**8) AutoPartOrderDetail (информация о деталях заказа)**

Foreign key: AutoPartID (идентификатор автозапчасти) из отношения AutoParts, OrderID (идентификатор заказа) из отношения AutoPartOrderHeader

Primary key: Составной из OrderID (идентификатор заказа), OrderDetailID (идентификатор детали заказа)

**9) Managers (информация о менеджерах)**

Primary key: ManagerID (идентификатор менеджера)

Foreign key: CarDealershipID (идентификатор филиала) из отношения CarDealerships.

**10) CustomerBase (информация о клиентах)**

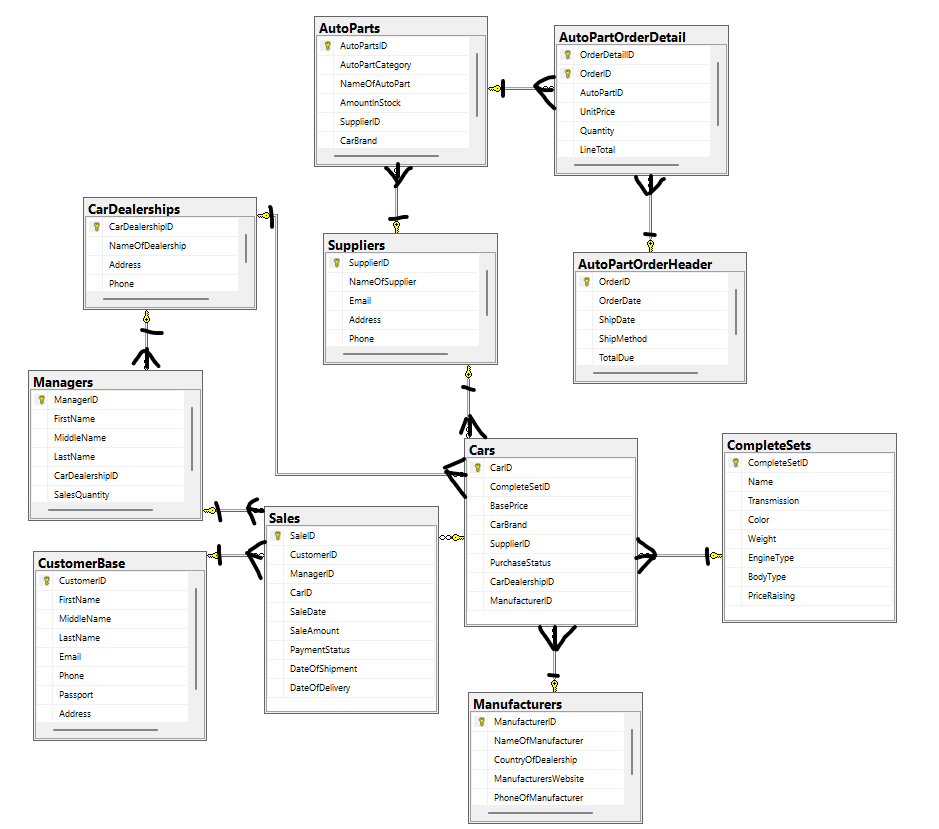
Primary key: CustomerID (идентификатор клиента)

**11) Sales (информация о продажах)**

Foreign key: CustomerID (идентификатор клиента) из отношения CustomerBase, ManagerID (идентификатор менеджера) из отношения Managers, CarID (идентификатор машины) из отношения Cars

Primary key: SaleID (идентификатор продажи)

**Графическое представление базы данных:**

Рис 1. Диаграмма базы данных

## **2.3 Анализ функциональных зависимостей**

**1) Cars (Информация о машинах)**

CarID → CompleteSetID, BasePrice, CarBrand, SupplierID, PurchaseStatus, CarDealershipID, ManufacturerID.

Идентификатор машины функционально определяет её комплектацию, модель, базовую стоимость, поставщика, филиал, в котором она находится, производителя и статуса покупки.

**2) CompleteSets ( Информация о комплектациях машин)**

CompleteSetID → Name, Transmission, Color, Weight, EngineType, BodyType, PriceRaising.

Идентификатор комплектации функционально определяет её название, коэффициент повышения цены, тип двигателя, кузова, трансмиссии, цвет и вес машины с данной комплектацией.

**3) Manufacturers (информация о производителях)**

ManufacturerID → NameOfManufacturer, CountryOfDealership, ManufacturersWebsite, PhoneOfManufacturer

Идентификатор производителя функционально определяет его полное название, страну производства, официальный сайт компании и номер телефона.

**4) CarDealerships (информация о филиалах)**

CarDealershipID → NameOfDealership, Address, Phone

Идентификатор филиала функционально определяет его полное название, адрес и номер телефона.

**5) Suppliers (информация о поставщиках)**

SupplierID → NameOfSupplier, Email, Address, Phone

Идентификатор поставщика функционально определяет его полное название, адрес и номер телефона.

**6) AutoParts (информация о автозапчастях)**

AutoPartsID → AutoPartCategory, NameOfAutoPart, AmountInStock, SupplierID, CarBrand

Идентификатор автозапчасти функционально определяет его полное название, категорию, количество на складе, поставщика и модель машины, которой эта запчасть подходит.

**7) AutoPartOrderHeader (информация о заказе)**

OrderID → OrderDate, ShipDate, ShipMethod, TotalDue

Идентификатор заказа функционально определяет его общую сумму, дату отправки и доставки.

**8) AutoPartOrderDetail (информация о деталях заказа)**

(OrderID, OrderDetailID) → OrderID, AutoPartID, UnitPrice, Quantity, LineTotal.

Комбинация идентификатора заказа и идентификатора детали заказа функционально определяет номер заказа, заказываемую запчасть, её количество в заказе, цену на единицу товара, и общую сумму по товару.

**9) Managers (информация о менеджерах)**

ManagerID → FirstName, MiddleName, LastName, CarDealershipID, SalesQuantity.

Идентификатор филиала функционально определяет его полное ФИО, место работы и количество заключённых сделок.

**10) CustomerBase (информация о клиентах)**

CustomerID → FirstName, MiddleName, LastName, Email, Phone, Passport, Address.

Идентификатор клиениа функционально определяет его полное ФИО, адрес электронной почты, номер телефона, паспортные данные и адрес.

**11) Sales (информация о продажах)**

SaleID → CustomerID, ManagerID, CarID, SaleDate, SaleAmount, PaymentStatus, DateOfShipment, DateOfDelivery

Идентификатор продажи функционально определяет клиента, менеджера и продаваемую машину, общую сумму продажи, статус оплаты и дату отправки и доставки.

Все отношения в базе данных находятся в 3-НФ, так как все атрибуты атомарные, нет повторяющихся групп, и все не ключевые атрибуты полностью зависят от первичного ключа, а также нет транзитивных зависимостей.

# **3. Заполнение базы данных информацией**

Для заполнения базы данных информацией я использовала ручное заполнение при помощи конструкции INSERT. Для генерации данных я использовала AI, а именно DeepSeek и ChatGPT 4-o.

Покажем заполнение БД информацией на нескольких примерах:  
**1) SQL запрос для заполнения отношения Manufacturers**

INSERT INTO Manufacturers (NameOfManufacturer, CountryOfDealership, ManufacturersWebsite, PhoneOfManufacturer)

VALUES

('Toyota', 'Japan', 'www.toyota.com', '+81-3-3817-7111'),

('Ford', 'USA', 'www.ford.com', '+1-800-392-3673'),

('Volkswagen', 'Germany', 'www.volkswagen.de', '+49-5361-9-0'),

('Honda', 'Japan', 'www.honda.com', '+81-3-3423-1111'),

('BMW', 'Germany', 'www.bmw.com', '+49-89-382-0'),

('Chevrolet', 'USA', 'www.chevrolet.com', '+1-800-222-1020'),

('Nissan', 'Japan', 'www.nissan-global.com', '+81-45-523-5523'),

('Mercedes-Benz', 'Germany', 'www.mercedes-benz.com', '+49-711-17-0'),

('Hyundai', 'South Korea', 'www.hyundai.com', '+82-2-3464-1111'),

('Tesla', 'USA', 'www.tesla.com', '+1-888-518-3752'),

('Renault', 'France', 'www.renault.com', '+33-1-76-84-80-00'),

('Audi', 'Germany', 'www.audi.com', '+49-841-89-0'),

('Kia', 'South Korea', 'www.kia.com', '+82-51-240-8114'),

('Subaru', 'Japan', 'www.subaru.com', '+81-3-3340-1111'),

('Mazda', 'Japan', 'www.mazda.com', '+81-3-5532-5801');

**2) SQL запрос для заполнения отношения CustomerBase**

INSERT INTO CustomerBase (FirstName, MiddleName, LastName, Email, Phone, Passport, Address)

VALUES

('Иван', 'Иванович', 'Иванов', 'ivan.ivanov@example.com', '+7-912-345-67-89', '4507 123456', 'г. Москва, ул. Ленина, д. 1'),

('Петр', 'Петрович', 'Петров', 'petr.petrov@example.com', '+7-912-345-67-90', '4508 654321', 'г. Санкт-Петербург, ул. Гагарина, д. 2'),

('Анна', 'Андреевна', 'Сидорова', 'anna.sidorova@example.com', '+7-912-345-67-91', '4509 112233', 'г. Новосибирск, ул. Кирова, д. 3'),

('Елена', 'Игоревна', 'Кузнецова', 'elena.kuznetsova@example.com', '+7-912-345-67-92', '4510 445566', 'г. Екатеринбург, ул. Мира, д. 4'),

('Дмитрий', 'Дмитриевич', 'Смирнов', 'dmitry.smirnov@example.com', '+7-912-345-67-93', '4511 778899', 'г. Казань, ул. Советская, д. 5'),

('Ольга', 'Олеговна', 'Морозова', 'olga.morozova@example.com', '+7-912-345-67-94', '4512 123123', 'г. Нижний Новгород, ул. Ленина, д. 6'),

('Сергей', 'Сергеевич', 'Васильев', 'sergey.vasilev@example.com', '+7-912-345-67-95', '4513 456456', 'г. Ростов-на-Дону, ул. Гагарина, д. 7'),

('Мария', 'Михайловна', 'Козлова', 'maria.kozlova@example.com', '+7-912-345-67-96', '4514 789789', 'г. Уфа, ул. Кирова, д. 8'),

('Алексей', 'Алексеевич', 'Новиков', 'alexey.novikov@example.com', '+7-912-345-67-97', '4515 321321', 'г. Красноярск, ул. Мира, д. 9'),

('Татьяна', 'Тимофеевна', 'Федорова', 'tatiana.fedorova@example.com', '+7-912-345-67-98', '4516 654654', 'г. Воронеж, ул. Советская, д. 10'),

('Владимир', 'Владимирович', 'Соколов', 'vladimir.sokolov@example.com', '+7-912-345-67-99', '4517 987987', 'г. Саратов, ул. Ленина, д. 11'),

('Ирина', 'Ивановна', 'Попова', 'irina.popova@example.com', '+7-912-345-68-00', '4518 123987', 'г. Омск, ул. Гагарина, д. 12'),

('Александр', 'Александрович', 'Лебедев', 'alexander.lebedev@example.com', '+7-912-345-68-01', '4519 456789', 'г. Краснодар, ул. Кирова, д. 13'),

('Наталья', 'Николаевна', 'Ковалева', 'natalia.kovaleva@example.com', '+7-912-345-68-02', '4520 789123', 'г. Тольятти, ул. Мира, д. 14'),

('Михаил', 'Михайлович', 'Орлов', 'mikhail.orlov@example.com', '+7-912-345-68-03', '4521 321789', 'г. Барнаул, ул. Советская, д. 15'),

('Екатерина', 'Евгеньевна', 'Алексеева', 'ekaterina.alekseeva@example.com', '+7-912-345-68-04', '4522 654123', 'г. Владивосток, ул. Ленина, д. 16'),

('Андрей', NULL, 'Семенов', 'andrey.semenov@example.com', '+7-912-345-68-05', '4523 987456', 'г. Ярославль, ул. Гагарина, д. 17'),

('Олег', NULL, 'Павлов', 'oleg.pavlov@example.com', '+7-912-345-68-06', '4524 123789', 'г. Иркутск, ул. Кирова, д. 18');

**3) SQL запрос для заполнения отношения Sales**

INSERT INTO Sales (CustomerID, ManagerID, CarID, PurchaseDate, PurchaseAmount, PaymentStatus, DateOfShipment, DateOfDelivery)

VALUES

(1, 1, 6, '2023-05-01', 55000.00, 1, '2023-10-01', '2023-15-01'), -- Toyota Supra Standard

(2, 2, 7, '2023-10-01', 42000.00, 1, '2023-15-01', '2023-20-01'), -- Toyota RAV4 Standard

(3, 3, 9, '2023-15-01', 31000.00, 1, '2023-20-01', '2023-25-01'), -- Ford Focus Sport

(4, 4, 11, '2023-20-01', 33000.00, 1, '2023-25-01', '2023-30-01'), -- Volkswagen Golf Sport

(5, 5, 14, '2023-25-01', 41000.00, 1, '2023-30-01', '2023-05-02'), -- Volkswagen Passat Sport

(6, 6, 16, '2023-05-02', 30000.00, 1, '2023-10-02', '2023-15-02'), -- Honda Civic Standard

(7, 7, 18, '2023-10-02', 57000.00, 1, '2023-15-02', '2023-20-02'), -- BMW 3 Series Luxury

(8, 8, 21, '2023-15-02', 37000.00, 1, '2023-20-02', '2023-25-02'), -- Honda CR-V Luxury

(9, 9, 23, '2023-20-02', 32000.00, 1, '2023-25-02', '2023-01-03'), -- Chevrolet Cruze Sport

(10, 10, 26, '2023-25-02', 26500.00, 1, '2023-01-03', '2023-06-03'), -- Nissan Sentra Luxury

(1, 1, 27, '2023-05-03', 38000.00, 1, '2023-10-03', '2023-15-03'), -- Nissan Rogue Luxury

(2, 2, 30, '2023-10-03', 50000.00, 1, '2023-15-03', '2023-20-03'), -- Chevrolet Camaro Luxury

(3, 3, 32, '2023-15-03', 21000.00, 1, '2023-20-03', '2023-25-03'), -- Nissan Sentra Standard

(4, 4, 33, '2023-20-03', 31000.00, 1, '2023-25-03', '2023-30-03'), -- Nissan Rogue Standard

(5, 5, 37, '2023-25-03', 48000.00, 1, '2023-30-03', '2023-05-04'), -- Nissan Rogue Platinum

(6, 6, 39, '2023-05-04', 35000.00, 1, '2023-10-04', '2023-15-04'), -- Toyota RAV4 Off-Road

(7, 7, 40, '2023-10-04', 33000.00, 1, '2023-15-04', '2023-20-04'), -- Ford Focus Titanium

(8, 8, 44, '2023-15-04', 41000.00, 1, '2023-20-04', '2023-25-04'), -- Volkswagen Passat Highline

(9, 9, 45, '2023-20-04', 65000.00, 1, '2023-25-04', '2023-30-04'), -- BMW 3 Series M-Sport

(10, 10, 46, '2023-25-04', 85000.00, 1, '2023-30-04', '2023-05-05'), -- BMW X5 M-Performance

(1, 1, 36, '2023-05-05', 23000.00, 1, '2023-10-05', '2023-15-05'), -- Chevrolet Cruze Standard

(2, 2, 35, '2023-10-05', 52000.00, 1, '2023-15-05', '2023-20-05'), -- Chevrolet Camaro Sport

(3, 3, 48, '2023-15-05', 29000.00, 1, '2023-20-05', '2023-25-05'), -- Nissan Sentra Sport

(4, 4, 49, '2023-20-05', 47000.00, 1, '2023-25-05', '2023-30-05'), -- Honda CR-V Off-Road

(5, 5, 50, '2023-25-05', 60000.00, 1, '2023-30-05', '2023-05-06') -- BMW X5 Off-Road

# **4. Описание представлений**

**Представление** — это объект базы данных Microsoft SQL Server, который хранит в себе запрос SELECT и в случае обращения к данному объекту будет возвращен результирующий набор данных, который формирует запрос, указанный в определении представления. Иными словами, это виртуальная (*логическая*) таблица, она не содержит в себе данных, но к ней можно обращаться как к обычной таблице, и она будет возвращать данные.

1. **Представление CarSalesDetails**

**Описание:** это представление создано, чтобы получать полную информацию о менеджере и клиенте, заключающих сделку. Оно показывает идентификатор сделки, купленную машину и общую сумму покупки, а также полную информацию о клиенте и менеджере, участвовавших в сделке, и филиал, в котором была заключена сделка.

**Структура:**

* SaleID – идентификатор сделки
* SaleDate – дата совершения продажи
* CarID – идентификатор покупаемой машины
* SaleAmount – сумма продажи
* PaymentStatus – статус оплаты
* CarDealership – филиал, в котором была заключена сделка
* Customer – полное ФИО клиента
* CustomerEmail – электронный адрес клинента
* CustomerPhone – номер телефона клиента
* CustomerPassport – паспортные данные клиента
* CustomerAddress – адрес клиента
* ManagerID – идентификатор менеджера
* Manager – полное ФИО менеджера

**Описание SQL кода:**

1)**SELECT –** выборка необходимых столбцов для представления

* s.SaleID,
* CAST(s.SaleDate AS date) AS SaleDate,
* s.CarID,
* s.SaleAmount,
* s.PaymentStatus,
* m.CarDealershipID AS CarDealership,
* (c.FirstName + ' ' + ISNULL(c.MiddleName,'') + ' ' + c.LastName) AS Customer,
* c.Email AS CustomerEmail,
* c.Phone AS CustomerPhone,
* c.Passport AS CustomerPassport,
* c.Address AS CustomerAddress,
* m.ManagerID,
* (m.FirstName + ' ' + ISNULL(m.MiddleName,'') + ' ' + m.LastName) AS Manager

2)**FROM и JOIN** - Использование двух таблиц для создания соединения.

* Sales – основная таблица с информацией о продаже
* CustomerBase – таблица для извлечения данных о клиенте
* Managers – таблица для извлечения данных о менеджере

3)**ISNULL** – функция, которое проверяет значение MiddleName на Null. Если оно равно Null, то MiddleName присваивается значение пустой строки – отчество отсутствует.

CREATE VIEW CarSalesDetails AS

SELECT

s.SaleID,

CAST(s.SaleDate AS date) AS SaleDate,

s.CarID,

s.SaleAmount,

s.PaymentStatus,

m.CarDealershipID AS CarDealership,

(c.FirstName + ' ' + ISNULL(c.MiddleName,'') + ' ' + c.LastName) AS Customer,

c.Email AS CustomerEmail,

c.Phone AS CustomerPhone,

c.Address AS CustomerAddress,

m.ManagerID,

(m.FirstName + ' ' + ISNULL(m.MiddleName,'') + ' ' + m.LastName) AS Manager

FROM

Sales s

JOIN

CustomerBase c ON s.CustomerID = c.CustomerID

JOIN

Managers m ON s.ManagerID = m.ManagerID;

GO

1. **Представление AvailableCarsForSale**

**Описание:** это представление создано, чтобы получить полную информацию о машинах, которые доступны к покупке. Оно показывает идентификатор машины, модель, итоговую стоимость, полную информацию о комплектации, а также информацию о производителе и салоне, в котором эта машина находится.

**Структура:**

* c.CarID – идентификатор машины
* c.CarBrand- модель машины
* (c.BasePrice \* cs.PriceRaising) AS Price – итоговая стоимость, найденная как произведение базовой цены на повышающий коэффициент
* cs.Name AS SetName – название комплектации
* cs.Transmission – тип трансмиссии
* cs.Color -цвет машины
* cs.BodyType -тип кузова
* cs.EngineType -тип двигателя
* cs.Weight – вес машины
* cd.NameOfDealership – название филиала
* cd.Address AS DealershipAddress – адрес филиала
* cd.Phone AS PhoneOfManufacturer – контактный номер филиала
* m.NameOfManufacturer – название производителя
* m.ManufacturersWebsite – официальный сайт производителя

**Описание SQL кода:**

1)**SELECT –** выборка необходимых столбцов для представления

* c.CarID,
* c.CarBrand,
* (c.BasePrice \* cs.PriceRaising) AS Price,
* cs.Name AS SetName,
* cs.Transmission,
* cs.Color,
* cs.BodyType,
* cs.EngineType,
* cs.Weight,
* cd.NameOfDealership,
* cd.Address AS DealershipAddress,
* cd.Phone AS PhoneOfManufacturer,
* m.NameOfManufacturer,
* m.ManufacturersWebsite

2)**FROM и JOIN** - Использование двух таблиц для создания соединения.

* Cars – основная таблица с информацией о машине
* CompleteSets – таблица для извлечения данных о комплектации для заданной машины
* Manufacturers– таблица для извлечения данных о производителе
* CarDealerships – таблица для извлечения данных о филиале

1. **WHERE** – фильтрация данных по статусу покупки равному 0, т.е машина ещё не куплена

CREATE VIEW AvailableCarsForSale AS

SELECT

c.CarID,

c.CarBrand,

(c.BasePrice \* cs.PriceRaising) AS Price,

cs.Name AS SetName,

cs.Transmission,

cs.Color,

cs.BodyType,

cs.EngineType,

cs.Weight,

cd.NameOfDealership,

cd.Address AS DealershipAddress,

cd.Phone AS PhoneOfManufacturer,

m.NameOfManufacturer,

m.ManufacturersWebsite

FROM

CarDealerships cd

JOIN

Cars c ON cd.CarDealershipID = c.CarDealershipID

JOIN

CompleteSets cs ON c.CompleteSetID = cs.CompleteSetID

JOIN

Manufacturers m ON m.ManufacturerID = c.ManufacturerID

WHERE

c.PurchaseStatus = 0

GO

**3) Представление ManagerRanking**

**Описание:** это представление создано для отображения рейтинга менеджеров на основе количества суммы всех сделок. Оно показывает идентификатор менеджера, название филиала, в котором он работает, полное ФИО сотрудника, количество совершённых сделок, их общую сумму и номер менеджера в рейтинге.

**Структура:**

* m.ManagerID – идентификатор менеджера
* cd.NameOfDealership – название филиала
* Manager – полное ФИО менеджера
* SalesQuantity – количество заключенных сделок
* TotalSalesAmount – общая сумма сделок
* ManagerRank - ранг менеджера по общей сумме сделок

**Описание SQL кода:**

1)**SELECT –** выборка необходимых столбцов для представления

m.ManagerID,

cd.NameOfDealership,

(m.FirstName + ' ' + ISNULL(m.MiddleName,'') + ' ' + m.LastName) AS Manager,

m.SalesQuantity,

SUM(COALESCE(s.SaleAmount, 0)) AS TotalSalesAmount,

DENSE\_RANK() OVER (ORDER BY SUM(s.SaleAmount) DESC) AS ManagerRank

2)**FROM и JOIN** - Использование двух таблиц для создания соединения.

Managers – основная таблица с информацией о менеджере

Sales– таблица для извлечения данных о продажах

Используем между ними **LEFT JOIN**, чтобы при соединение таблицы включались все менеджера, даже те, у которых нет родаж

CarDealerships– таблица для извлечения данных о филиале

3)  **SUM(…) –** агрегатная функция для подсчета общей суммы продаж.

4) **DENSE\_RANK() OVER (PARTITION BY cd.NameOfDealership ORDER BY SUM(s.SaleAmount) DESC)–** функция присваивает ранг каждому менеджеру в зависимости от общей суммы их продаж. Partition by cd.NameOfDealership разбивает менеджеров на подгруппы в зависимости от филиала, в котором они работают. Ранги присваиваются по порядку внутри каждой подгруппы. Сортировка выполняется по убыванию этой суммы. Если два менеджера имеют одинаковую общую сумму продаж, то им присваивается одинаковый ранг, который продолжается у следующего по списку менеджера без пропуска номера.

5) **ISNULL** – функция, которое проверяет значение MiddleName на Null. Если оно равно Null, то MiddleName присваивается значение пустой строки – отчество отсутствует.

6) **GROUP BY** – группировка данных по заданным параметрам для корректной работы агретных функций SUM(), DENSE\_RANK().

7) **COALESCE(s.SaleAmount, 0) –** заменяет NULL на 0.

Данное представление может быть полезно, например, для назначения премии для сотрудников с лучшими показателями труда в каждом отдельном автосалоне.

CREATE VIEW ManagerRanking AS

SELECT

m.ManagerID,

cd.NameOfDealership,

(m.FirstName + ' ' + ISNULL(m.MiddleName,'') + ' ' + m.LastName) AS Manager,

SUM(COALESCE(s.SaleAmount, 0)) AS TotalSalesAmount,

DENSE\_RANK() OVER (PARTITION BY m.CarDealershipID ORDER BY SUM(s.SaleAmount) DESC) AS ManagerRank

FROM

Managers m

LEFT JOIN

Sales s ON m.ManagerID = s.ManagerID

JOIN

CarDealerships cd ON m.CarDealershipID = cd.CarDealershipID

GROUP BY

m.ManagerID, m.FirstName, m.MiddleName, m.LastName, cd.NameOfDealership, m.CarDealershipID;

GO

1. **Представение AutoPartOrderStatus**

**Описание:** это представление создано для отображения статуса доставки заказа автозапчастей. Оно показывает идентификатор заказа, метод доставки, общую сумму заказа и статус доставки: «Доставлен», «Доставляется», «Оформляется».

Данное представление поддерживает вставку и обновление данных благодаря триггерам Update\_OrderHeader и Insert\_OrderHeader.

**Структура:**

* OrderID – идентификатор заказа
* ShipMethod – метод/ способ доставки
* TotalDue – общая сумма заказа
* OrderDate - дата оформления заказа
* ShipDate – ожидаемая дата доставки
* OrderStatus – статус доставки

**Описание SQL кода:**

1. **SELECT –** выборка необходимых столбцов для представления

oh.OrderID,

oh.ShipMethod,

oh.TotalDue,

CAST(oh.OrderDate AS DATE) AS OrderDate ,

CAST(oh.ShipDate AS DATE) AS ShipDate,

CASE…END AS OrderStatus

1. **FROM и JOIN** - Использование двух таблиц для создания соединения.

AutoPartOrderHeader– основная таблица с информацией о заказе автозапчастей

1. **CASE … -** условный оператор, позволяющий определять статус заказа на основе дат заказа и доставки, а также текущей даты, получаем с помощью функции GETDATE().

CREATE VIEW AutoPartOrderStatus

AS

SELECT

oh.OrderID,

oh.ShipMethod,

oh.TotalDue,

CAST(oh.OrderDate AS DATE) AS OrderDate ,

CAST(oh.ShipDate AS DATE) AS ShipDate,

CASE

WHEN ShipDate <= GETDATE() THEN N'Доставлено'

WHEN OrderDate < GETDATE() AND oh.ShipDate > GETDATE() THEN N'Доставляется'

WHEN OrderDate = GETDATE() THEN N'Оформляется'

END AS OrderStatus

FROM AutoPartOrderHeader as oh

GO

1. **Представление AvailableCompleteSetsForCar**

**Описание:** это представление отображает все возможные комплектации для каждой модели машин. Оно показывает модель машины, название комплектации и итоговую стоимость машины с выбранной комплектацией.

**Структура:**

* CarBrand – название модели машины
* CompleteSetID – идентификатор комплектации машины
* Name – название комплектации
* TotalPrice – итоговая стоимость машины с выбранной комплектацией

**Описание SQL кода:**

1. **SELECT –** выборка необходимых столбцов для представления

c.CarBrand,

c.CompleteSetID,

cs.Name,

(c.BasePrice \* cs.PriceRaising) AS TotalPrice

1. **FROM и JOIN** - Использование двух таблиц для создания соединения.

* Cars – основная таблица с информацией о машинах (содержит модели машин)
* CompleteSets – таблица для извлечения информации о комплектациях машин

1. **ORDER BY** – сортировка, необходимая для выведения всех комплектаций для каждой модели машины подряд.
2. **TOP 100 PERCENT –** выбираем все строки (100% от общего количества). Позволяет использовать сортировку в представлении.

CREATE VIEW AvailableCompleteSetsForCar

AS

SELECT TOP 100 PERCENT

c.CarBrand,

c.CompleteSetID,

cs.Name,

(c.BasePrice \* cs.PriceRaising) AS TotalPrice

FROM

Cars c

JOIN

CompleteSets cs

ON

cs.CompleteSetID = c.CompleteSetID

ORDER BY

c.CarBrand, c.CompleteSetID;

GO

# **Описание функций**

Функция в SQL Server — это хранимая программа, в которую можно передавать параметры и возвращать значение.

**1) Функция GetTopCustomer**

**Описание:** определяет клиента с самой большой общей суммой покупок.

**Описание SQL кода:**

1. **RETURNS TABLE** – функция возвращает табличное значение
2. **SELECT TOP 1** – выборка одной записи с наибольшей общей суммой покупок
3. **SUM(s.SaleAmount)** – агрегатная функция для суммирования всех покупок клиента
4. **ORDER BY TotalSaleAmount DESC** – сортировка по убыванию общей суммы покупок
5. **GROUP BY…** - группировка данных по каждому клиенту для корректной работы функции SUM()
6. **ISNULL** – функция, которое проверяет значение MiddleName на Null. Если оно равно Null, то MiddleName присваивается значение пустой строки – отчество отсутствует.

CREATE FUNCTION GetTopCustomer()

RETURNS TABLE

AS

RETURN (

SELECT TOP 1

cb.CustomerID,

(cb.FirstName + ' ' + ISNULL(cb.MiddleName,'') + ' '+ cb.LastName) AS Customer,

cb.Email,

cb.Phone,

cb.Address,

SUM(s.SaleAmount) AS TotalSaleAmount

FROM

CustomerBase cb

JOIN

Sales s ON cb.CustomerID = s.CustomerID

GROUP BY

cb.CustomerID, cb.FirstName, cb.LastName, cb.MiddleName, cb.Email, cb.Phone, cb.Address

ORDER BY

TotalSaleAmount DESC

);

GO

**2) Функция GetMonthlySalesSums**

**Описание:** вычисляет общую сумму продаж за месяц во всех филиалах автосалона за указанный год. На вход принимает значение типа INT – номер года.

**Описание SQL кода:**

1. **RETURNS TABLE** – функция возвращает табличное значение
2. **@Year INT** – входной параметр функции, год, по которому ведётся расчет сумм.
3. **INSERT INTO @ResultTable** – вставка данных из select-запрос в табличную переменную, которую функция возвращает
4. **DATENAME(MONTH, SaleDate)** – возвращает название месяца
5. **MONTH(SaleDate)** – возвращает порядковый номер месяца
6. **SUM(SaleAmount)** – функция, суммирующая продажи по месяцам и году
7. **WHERE YEAR(SaleDate) = @Year** – фильтрация данных на соответствие заданному году
8. **GROUP BY MONTH(SaleDate), DATENAME(MONTH, SaleDate)** – группировка для корректной работы агрегатной функции SUM()

CREATE FUNCTION GetMonthlySalesSums (@Year INT)

RETURNS @ResultTable TABLE (

Year INT,

MonthName NVARCHAR(20),

MonthNumber INT,

TotalAmount DECIMAL(18, 2)

)

AS

BEGIN

INSERT INTO @ResultTable

SELECT

@Year AS Year,

DATENAME(MONTH, SaleDate) AS MonthName,

MONTH(SaleDate) AS MonthNumber,

SUM(SaleAmount) AS TotalAmount

FROM

Sales

WHERE

YEAR(SaleDate) = @Year

GROUP BY

MONTH(SaleDate), DATENAME(MONTH, SaleDate);

RETURN;

END;

GO

**3) Функция SumSalesForYear**

**Описание:** вычисляет общую сумму продаж за год в указанном филиале автосалона. На вход принимает два параметра: идентификатор филиала и номер года.

**Описание SQL кода:**

1. **RETURNS DECIMAL** – функция возвращает число с плавающей точкой с точностью 18 знаков и одним знаком после запятой.
2. **@Year INT** – входной параметр функции, год, по которому ведётся расчет суммы.
3. **@DealershipID INT** – входной параметр функции, идентификатор автосалона.
4. **DECLARE @TotalSales DECIMAL(18, 1)** – объявление переменной для хранения значения суммы.
5. **SUM(SaleAmount)** – функция, суммирующая продажи.
6. **ISNULL(@TotalSales, 0.0)** – функция, возвращающая значение 0.0, если в @TotalSales содержится NULL.

CREATE FUNCTION SumSalesForYear(@DealershipID INT, @Year INT)

RETURNS DECIMAL(18, 1)

AS

BEGIN

DECLARE @TotalSales DECIMAL(18, 1);

SELECT @TotalSales = SUM(s.SaleAmount)

FROM Sales s

JOIN Cars c ON s.CarID = c.CarID

WHERE c.CarDealershipID = @DealershipID

AND YEAR(s.SaleDate) = @Year;

RETURN ISNULL(@TotalSales, 0.0);

END;

GO

# **Описание хранимых процедур**

Хранимая процедура (Stored Procedure) в MS SQL Server — это набор инструкций SQL, которые хранятся на сервере базы данных и могут быть выполнены как единое целое. Хранимые процедуры используются для выполнения определенных операций, таких как вставка, обновление, удаление данных, выполнение сложных запросов и других задач.

1. **Процедура CompareCars**

**Описание:** данная процедура предназначена для сравнения характеристик двух выбранных машин.

**Подробное описание:**

1. **Параметры процедуры:** процедура принимает на вход идентификаторы двух машин, характеристики которых необходимо сравнить.
2. **Транзакция:** Процедура начинает транзакцию с помощью BEGIN TRANSACTION, чтобы гарантировать целостность данных
3. **Проверка существования машин:** Процедура проверяет, существуют ли машины с указанными CarID. Если машина не существует, выполняется откат транзакции.
4. **Проверка различия вводимых CarID:** Процедура проверяет, совпадают ли введённые CarID. Если они совпадают, выполняется откат транзакции.
5. **DECLARE @Comparison TABLE –** объявляется табличная переменная для хранения результата процедуры.
6. **Создание временных таблиц –** для сохранения характеристик сравниваемых машин.
7. **Выборка данных о машинах во временные таблицы.** При этом применяется сцепление результирующих наборов с помощью UNION ALL.
8. **Использование курсора.** Курсор применяется для перебора строк из предыдущего пункта и последующего сравнения характеристик с применением конструкции CASE… .
9. **Фиксация транзакции** при успешном её завершении.
10. **Обработка ошибок.** Если произошла ошибка, то выводится сообщение с указанием её текста и номера.

CREATE PROCEDURE CompareCars

@CarID1 INT,

@CarID2 INT

AS

BEGIN

BEGIN TRY

BEGIN TRANSACTION;

IF NOT EXISTS (SELECT 1 FROM Cars WHERE CarID = @CarID1)

BEGIN

DECLARE @CarError1 NVARCHAR(100) = N'Машины с указанным CarID = '+ CAST(@CarID1 AS nvarchar) + N' не существует';

THROW 80001, @CarError1, 1;

END

IF NOT EXISTS (SELECT 1 FROM Cars WHERE CarID = @CarID2)

BEGIN

DECLARE @CarError2 NVARCHAR(100) = N'Машины с указанным CarID = '+ CAST(@CarID2 AS nvarchar) + N' не существует';

THROW 80002, @CarError2, 1;

END

IF @CarID1 = @CarID2

BEGIN

DECLARE @SameCarError NVARCHAR(100) = N'ID машин должны быть различными.';

THROW 80003, @SameCarError, 1;

END

DECLARE @Comparison TABLE (

Attribute NVARCHAR(50),

Value NVARCHAR(100)

);

SELECT

c.CarID,

c.CarBrand,

c.BasePrice,

cs.Transmission,

cs.Color,

cs.Weight,

cs.EngineType,

cs.BodyType,

c.BasePrice \* cs.PriceRaising AS TotalPrice

INTO #Car1

FROM Cars c

JOIN CompleteSets cs ON c.CompleteSetID = cs.CompleteSetID

WHERE c.CarID = @CarID1;

SELECT

c.CarID,

c.CarBrand,

c.BasePrice,

cs.Transmission,

cs.Color,

cs.Weight,

cs.EngineType,

cs.BodyType,

c.BasePrice \* cs.PriceRaising AS TotalPrice

INTO #Car2

FROM Cars c

JOIN CompleteSets cs ON c.CompleteSetID = cs.CompleteSetID

WHERE c.CarID = @CarID2;

DECLARE @Attribute NVARCHAR(50), @Value1 NVARCHAR(100), @Value2 NVARCHAR(100);

DECLARE AttributeCursor CURSOR FOR

SELECT 'CarBrand', c1.CarBrand, c2.CarBrand FROM #Car1 c1, #Car2 c2

UNION ALL

SELECT 'Transmission', c1.Transmission, c2.Transmission FROM #Car1 c1, #Car2 c2

UNION ALL

SELECT 'Color', c1.Color, c2.Color FROM #Car1 c1, #Car2 c2

UNION ALL

SELECT 'Weight', CAST(c1.Weight AS NVARCHAR), CAST(c2.Weight AS NVARCHAR) FROM #Car1 c1, #Car2 c2

UNION ALL

SELECT 'EngineType', c1.EngineType, c2.EngineType FROM #Car1 c1, #Car2 c2

UNION ALL

SELECT 'BodyType', c1.BodyType, c2.BodyType FROM #Car1 c1, #Car2 c2

UNION ALL

SELECT 'TotalPrice', CAST(c1.TotalPrice AS NVARCHAR), CAST(c2.TotalPrice AS NVARCHAR) FROM #Car1 c1, #Car2 c2;

OPEN AttributeCursor;

FETCH NEXT FROM AttributeCursor INTO @Attribute, @Value1, @Value2;

WHILE @@FETCH\_STATUS = 0

BEGIN

INSERT INTO @Comparison (Attribute, Value)

SELECT @Attribute,

CASE

WHEN @Value1 = @Value2 THEN 'Same ' + @Attribute + ': ' + @Value1

ELSE @Value1 + ' vs ' + @Value2

END;

FETCH NEXT FROM AttributeCursor INTO @Attribute, @Value1, @Value2;

END

CLOSE AttributeCursor;

DEALLOCATE AttributeCursor;

SELECT \* FROM @Comparison;

DROP TABLE #Car1;

DROP TABLE #Car2;

COMMIT TRANSACTION;

END TRY

BEGIN CATCH

IF @@TRANCOUNT > 0 ROLLBACK TRANSACTION;

DECLARE @ErrorMessage NVARCHAR(100);

SET @ErrorMessage = ERROR\_MESSAGE();

DECLARE @ErrorNumber INT;

SET @ErrorNumber = ERROR\_NUMBER();

THROW @ErrorNumber, @ErrorMessage, 1;

END CATCH;

END;

GO

**2) Процедура AddSale**

**Описание:** данная процедура предназначена для добавления новой продажи в таблицу Sales.

**Подробное описание:**

1. **Параметры процедуры:** процедура принимает на вход идентификаторы клиента, менеджера и продаваемой машины, а также дату продажи, статус оплаты, дату отправки и доставки.
2. **Транзакция:** Процедура начинает транзакцию с помощью BEGIN TRANSACTION, чтобы гарантировать целостность данных
3. **Проверка существования машин:** Процедура проверяет, существуют ли машины с указанными CarID. Если машина не существует, выполняется откат транзакции.
4. **Проверка возможности покупки машины:** Процедура проверяет, можно ли купить данную машину. Если её PurchaseStatus равен 1, то машина уже куплена, и выполняется откат транзакции. Если он равен 0, то машина не продана и готова к покупке.
5. **Проверка совпадения CarDealershipID у машины и менеджера:** менеджер может продавать машины только из того автосалона, в котором работает. Поэтому в случаем несовпадения CarDealershipIDвыполняется откат транзакции.
6. **Проверка правильности введения дат отправки и доставки.** Дата доставки должна быть больше даты отправки, в противном случае выполняется откат транзакции.
7. **Вычисление общей суммы продажи.** Рассчитывается как произведение BasePrice и PriceRaising для продаваемой машины.
8. **Вставка данных в Sales.**
9. **Обработка ошибок.** Если произошла ошибка, то выводится сообщение с указанием её текста и номера.

CREATE PROCEDURE AddSale

@CustomerID INT,

@ManagerID INT,

@CarID INT,

@SaleDate DATETIME,

@PaymentStatus BIT,

@DateOfShipment DATETIME,

@DateOfDelivery DATETIME

AS

BEGIN

BEGIN TRY

BEGIN TRANSACTION;

IF NOT EXISTS (SELECT 1 FROM Cars WHERE CarID = @CarID)

BEGIN

DECLARE @CarError NVARCHAR(100) = N'Машины с указанным CarID = '+ CAST(@CarID AS nvarchar) + N' не существует';

THROW 50004, @CarError, 1;

END

DECLARE @PurchaseStatus BIT;

SELECT @PurchaseStatus = PurchaseStatus FROM Cars WHERE CarID = @CarID;

IF @PurchaseStatus = 1

BEGIN

DECLARE @PurchaseStatusError NVARCHAR(100) = N'Машина с указанным CarID = '+ CAST(@CarID AS nvarchar) + N' уже продана';

THROW 50005, @PurchaseStatusError, 1;

END

DECLARE @ManagerDealershipID INT;

DECLARE @CarDealershipID INT;

SELECT @ManagerDealershipID = CarDealershipID FROM Managers WHERE ManagerID = @ManagerID;

SELECT @CarDealershipID = CarDealershipID FROM Cars WHERE CarID = @CarID;

IF @ManagerDealershipID <> @CarDealershipID

BEGIN

DECLARE @DealershipError NVARCHAR(100) = N'Менеджер с ManagerID = ' + CAST(@ManagerID AS NVARCHAR) +

N' не может продавать машину с CarID = ' + CAST(@CarID AS NVARCHAR) +

N' из другого автосалона (CarDealershipID не совпадают)';

THROW 50007, @DealershipError, 1;

END

IF @DateOfShipment >= @DateOfDelivery

BEGIN

DECLARE @DateError NVARCHAR(100) = 'Дата отправки должна быть меньше даты доставки';

THROW 50006, @DateError, 1;

END

DECLARE @SaleAmount DECIMAL(18,1);

DECLARE @BasePrice DECIMAL(18,1);

DECLARE @PriceRaising DECIMAL(18,1);

SELECT @BasePrice = BasePrice FROM Cars WHERE CarID = @CarID;

SELECT @PriceRaising = PriceRaising FROM

Cars as c

JOIN CompleteSets as cs

ON c.CompleteSetID = cs.CompleteSetID

SET @SaleAmount = @BasePrice \* ISNULL(@PriceRaising, 1);

INSERT INTO Sales (CustomerID, ManagerID, CarID, SaleDate, SaleAmount, PaymentStatus, DateOfShipment, DateOfDelivery)

VALUES (@CustomerID, @ManagerID, @CarID, @SaleDate, @SaleAmount, @PaymentStatus, @DateOfShipment, @DateOfDelivery);

COMMIT TRANSACTION;

END TRY

BEGIN CATCH

IF @@TRANCOUNT > 0 ROLLBACK TRANSACTION;

DECLARE @ErrorMessage NVARCHAR(100);

SET @ErrorMessage = ERROR\_MESSAGE();

DECLARE @ErrorNumber INT;

SET @ErrorNumber = ERROR\_NUMBER();

THROW @ErrorNumber, @ErrorMessage, 1;

END CATCH;

END;

GO

**3) Процедура AddNewCustomer**

**Описание:** данная процедура предназначена для добавления нового покупателя в базу клиентов автосалона.

**Подробное описание:**

1. **Параметры процедуры:** процедура принимает на вход имя, отчество и фамилию клиента, а также его адрес электронной почты, номер телефона, паспортные данные и адрес.
2. **Транзакция:** Процедура начинает транзакцию с помощью BEGIN TRANSACTION, чтобы гарантировать целостность данных
3. **Проверка единственности клиента.** В клиентской базе не должно содержаться различных записей с информацией об одном и том же клиенте, поэтому перед вставкой проверяем уникальные данные клиента, такие как адрес электронной почты, номер телефона, паспортные данные, чтобы избежать повторного внесения информации о нём в CustomerBase.
4. **Вставка данных в CustomerBase.**
5. **Обработка ошибок.** Если произошла ошибка, то выводится сообщение с указанием её текста и номера.

CREATE PROCEDURE AddNewCustomer

@FirstName NVARCHAR(100),

@MiddleName NVARCHAR(100),

@LastName NVARCHAR(100),

@Email NVARCHAR(100),

@Phone NVARCHAR(20),

@Passport NVARCHAR(50),

@Address NVARCHAR(200)

AS

BEGIN

BEGIN TRY

BEGIN TRANSACTION;

IF EXISTS (SELECT 1 FROM CustomerBase WHERE Email = @Email)

BEGIN

DECLARE @EmailError NVARCHAR(100);

SET @EmailError = N'Клиент c Email ' + @Email + N' уже существует в клиентской базе';

THROW 50001, @EmailError, 1;

RETURN;

END

IF EXISTS (SELECT 1 FROM CustomerBase WHERE Passport = @Passport)

BEGIN

DECLARE @PassportError NVARCHAR(100);

SET @PassportError = N'Клиент c паспортными данными ' + @Passport + N' уже существует в клиентской базе';

THROW 50002, @PassportError, 1;

RETURN;

END

IF EXISTS (SELECT 1 FROM CustomerBase WHERE Phone = @Phone)

BEGIN

DECLARE @PhoneError NVARCHAR(100);

SET @PhoneError = N'Клиент c номером телефона ' + @Phone + N' уже существует в клиентской базе';

THROW 50003, @PhoneError, 1;

RETURN;

END

INSERT INTO CustomerBase (FirstName, MiddleName, LastName, Email, Phone, Passport, Address)

VALUES (@FirstName, @MiddleName, @LastName, @Email, @Phone, @Passport, @Address);

COMMIT TRANSACTION;

END TRY

BEGIN CATCH

IF @@TRANCOUNT > 0 ROLLBACK TRANSACTION;

DECLARE @ErrorMessage NVARCHAR(100);

SET @ErrorMessage = ERROR\_MESSAGE();

DECLARE @ErrorNumber INT;

SET @ErrorNumber = ERROR\_NUMBER();

THROW @ErrorNumber, @ErrorMessage, 1;

END CATCH;

END;

GO

**4) Процедура CreateAutoPartOrderHeader**

**Описание:** данная процедура предназначена для создания нового заказа на автозапчасти.

**Подробное описание:**

1. **Параметры процедуры:** процедура принимает на вход дату заказа, дату доставки и способ доставки.
2. **Транзакция:** Процедура начинает транзакцию с помощью BEGIN TRANSACTION, чтобы гарантировать целостность данных
3. **Проверка правильности введения дат отправки и доставки.** Дата доставки должна быть больше даты отправки, в противном случае выполняется откат транзакции.
4. **Вставка данных вAutoPartOrderHeader.**
5. **Возвращение идентификатора созданного заказа.**
6. **Обработка ошибок.** Если произошла ошибка, то выводится сообщение с указанием её текста и номера.

CREATE PROCEDURE CreateAutoPartOrderHeader

@OrderDate DATETIME,

@ShipDate DATETIME,

@ShipMethod NVARCHAR(100)

AS

BEGIN

BEGIN TRY

BEGIN TRANSACTION;

IF @ShipDate <= @OrderDate

BEGIN

DECLARE @DateError NVARCHAR(100) = N'Дата доставки должна быть больше даты отправки.';

THROW 60001, @DateError, 1;

END;

INSERT INTO AutoPartOrderHeader (OrderDate, ShipDate, ShipMethod, TotalDue)

VALUES (@OrderDate, @ShipDate, @ShipMethod, 0);

COMMIT TRANSACTION;

DECLARE @NewOrderID INT = SCOPE\_IDENTITY();

RETURN @NewOrderID;

END TRY

BEGIN CATCH

IF @@TRANCOUNT > 0 ROLLBACK TRANSACTION;

DECLARE @ErrorMessage NVARCHAR(100);

SET @ErrorMessage = ERROR\_MESSAGE();

DECLARE @ErrorNumber INT;

SET @ErrorNumber = ERROR\_NUMBER();

THROW @ErrorNumber, @ErrorMessage, 1;

END CATCH;

END;

GO

1. **Процедура AddAutoPartOrderDetail**

**Описание:** данная процедура позволяет добавлять детали заказа на автозапчасти, а именно какую деталь/ детали и в каком количестве необходимо заказать.

**Подробное описание:**

1. **Параметры процедуры:** процедура принимает на вход идентификатор заказа, идентификатор заказываемой детали, цену за единицу детали и количество.
2. **Транзакция:** Процедура начинает транзакцию с помощью BEGIN TRANSACTION, чтобы гарантировать целостность данных
3. **Проверка существования заказа.**
4. **Проверка возможности внесения деталей в заказ.** Если текущая дата не совпадает с датой заказа, то детали в него внести уже нельзя, потому что он передан в доставку.
5. **Проверка, что указанное количество деталей больше 0.**
6. **Проверка, что указанная цена больше 0 и не NULL.**
7. **Вставка данных вAutoPartOrderDetail.**
8. **Обработка ошибок.** Если произошла ошибка, то выводится сообщение с указанием её текста и номера.

CREATE PROCEDURE AddAutoPartOrderDetail

@OrderID INT,

@AutoPartID INT,

@UnitPrice DECIMAL(10, 2),

@Quantity INT

AS

BEGIN

BEGIN TRY

BEGIN TRANSACTION;

IF NOT EXISTS (SELECT 1 FROM AutoPartOrderHeader WHERE OrderID = @OrderID)

BEGIN

DECLARE @OrderIDError NVARCHAR(100) = N'Заказ с указанным OrderID = ' + CAST(@OrderID AS NVARCHAR) + N' не существует.';

THROW 70002, @OrderIDError, 1;

END;

DECLARE @OrderDate DATETIME;

SELECT @OrderDate = OrderDate FROM AutoPartOrderHeader WHERE OrderID = @OrderID;

IF @OrderDate <> CAST(GETDATE() AS DATE)

BEGIN

DECLARE @DateError NVARCHAR(100) = N'Заказ уже передан в доставку. Нельзя добавлять детали к заказу, если дата заказа не совпадает с текущей датой.';

THROW 70005, @DateError, 1;

END;

IF @Quantity <= 0 OR @Quantity IS NULL

BEGIN

DECLARE @QuantityError NVARCHAR(100) = N'Количество должно быть больше нуля и не может быть NULL..';

THROW 70003, @QuantityError, 1;

END;

IF @UnitPrice <= 0 OR @UnitPrice IS NULL

BEGIN

DECLARE @PriceError NVARCHAR(100) = N'Цена должна быть больше нуля и не может быть NULL.';

THROW 70004, @PriceError, 1;

END;

INSERT INTO AutoPartOrderDetail (OrderID, AutoPartID, UnitPrice, Quantity, LineTotal)

VALUES (@OrderID, @AutoPartID, @UnitPrice, @Quantity, 0);

COMMIT TRANSACTION;

END TRY

BEGIN CATCH

IF @@TRANCOUNT > 0 ROLLBACK TRANSACTION;

DECLARE @ErrorMessage NVARCHAR(100);

SET @ErrorMessage = ERROR\_MESSAGE();

DECLARE @ErrorNumber INT;

SET @ErrorNumber = ERROR\_NUMBER();

THROW @ErrorNumber, @ErrorMessage, 1;

END CATCH;

END;

GO

# **Описание триггеров**

Триггер (Trigger) в MS SQL Server — это специальный тип хранимой процедуры, которая автоматически выполняется при наступлении определенного события, такого как вставка, обновление или удаление данных в таблице. Триггеры используются для обеспечения целостности данных, выполнения дополнительных операций или реагирования на изменения в базе данных.

1. **Триггер UpdateCarPurchaseStatus**

**Описание:** данный триггер предназначен для изменения статуса покупки машины PurchaseStatus на 1 после внесение данных о продаже этой машины в Sales.

**Подробное описание:**

1. **Триггер срабатывает после вставки.** Этот триггер активируется после вставки новых строк в отношение Sales.
2. **SET NOCOUNT ON:** устанавливает вариант NOCOUNT для предотвращения возвращения счетчика строк.
3. **Обновление значения атрибута PurchaseStatus.** После покупки машины атрибуту PurchaseStatus присваивается значение 1.

CREATE TRIGGER UpdateCarPurchaseStatus

ON Sales

AFTER INSERT

AS

BEGIN

SET NOCOUNT ON;

UPDATE Cars

SET PurchaseStatus = 1

WHERE CarID IN (SELECT CarID FROM inserted);

END;

GO

1. **Триггер UpdateManagerSalesQuantity**

**Описание:** данный триггер предназначен для увеличения количества заключенных сделок для менеджера, который проводил сделку, внесённую в Sales.

**Подробное описание:**

1. **Триггер срабатывает после вставки.** Этот триггер активируется после вставки новых строк в отношение Sales.
2. **SET NOCOUNT ON:** устанавливает вариант NOCOUNT для предотвращения возвращения счетчика строк.
3. **Обновление количества заключённых сделок для менеджера.** После покупки машины значение атрибута SalesQuantity в отношении Managers для выбранного менеджера увеличивается на 1.

CREATE TRIGGER UpdateManagerSalesQuantity

ON Sales

AFTER INSERT

AS

BEGIN

SET NOCOUNT ON;

UPDATE Managers

SET SalesQuantity = SalesQuantity + 1

FROM Managers

INNER JOIN inserted ON Managers.ManagerID = inserted.ManagerID;

END;

GO

1. **Триггер CalculateLineTotal**

**Описание:** данный триггер предназначен для автоматического подсчёта суммы заказа по заказываемой детали в отношении AutoPartOrderDetail.

**Подробное описание:**

1. **Триггер срабатывает после вставки.** Этот триггер активируется после вставки новых строк в отношение AutoPartOrderDetail.
2. **SET NOCOUNT ON:** устанавливает вариант NOCOUNT для предотвращения возвращения счетчика строк.
3. **Обновление суммы заказа по детали.** После внесения новых данных в AutoPartOrderDetail, значение суммы заказа по детали автоматически высчитывается как произведение UnitPrice на Quantity.

CREATE TRIGGER CalculateLineTotal

ON AutoPartOrderDetail

AFTER INSERT, UPDATE

AS

BEGIN

SET NOCOUNT ON;

UPDATE apod

SET apod.LineTotal = apod.UnitPrice \* apod.Quantity

FROM AutoPartOrderDetail apod

INNER JOIN inserted i ON apod.OrderID = i.OrderID AND apod.OrderDetailID = i.OrderDetailID;

END;

1. **Триггер CalculateTotalDue**

**Описание:** данный триггер создан для автоматического подсчёта общей суммы заказа при внесении в него новых деталей.

**Подробное описание:**

1. **Триггер срабатывает после вставки.** Этот триггер активируется после вставки новых строк в отношение AutoPartOrderDetail.
2. **SET NOCOUNT ON:** устанавливает вариант NOCOUNT для предотвращения возвращения счетчика строк.
3. **Обновление общей суммы заказа.** После внесения новых данных в AutoPartOrderDetail, значение общей суммы закзаза автоматически высчитывается как сумма всех LineTotal (сумм по детали) из AutoPartOrderDetail.
4. Функция **COALESCE** используется для замены NULL на 0, если отсутствует сумма по детали в AutoPartOrderDetail.

CREATE TRIGGER CalculateTotalDue

ON AutoPartOrderDetail

AFTER INSERT, UPDATE, DELETE

AS

BEGIN

SET NOCOUNT ON;

UPDATE apoh

SET apoh.TotalDue = (

SELECT COALESCE(SUM(apod.LineTotal), 0)

FROM AutoPartOrderDetail apod

WHERE apod.OrderID = apoh.OrderID

)

FROM AutoPartOrderHeader apoh

WHERE apoh.OrderID IN (SELECT DISTINCT OrderID FROM inserted UNION SELECT DISTINCT OrderID FROM deleted);

END;

GO

1. **Триггер InsertOrderHeader**

**Описание:** данный триггер вызывается вместо вставки данных в представление AutoPartOrderStatus. Данный триггер обрабатывает данные поступающие на вставку таким образом, чтобы добавить новые данные о заказе в отношение AutoPartOrderHeader.

**Подробное описание:**

1. **Триггер срабатывает вместо вставки.** Этот триггер срабатывает вместо операции вставки INSERT в представление AutoPartOrderStatus.
2. **SET NOCOUNT ON:** устанавливает вариант NOCOUNT для предотвращения возвращения счетчика строк.
3. **Проверка, что вносятся значения только для допустимых атрибутов.**
4. **Проверка правильности вносимых данных.** Дата доставки должна быть больше даты заказа.

CREATE TRIGGER InsertOrderHeader

ON AutoPartOrderStatus

INSTEAD OF INSERT

AS

BEGIN

SET NOCOUNT ON;

IF EXISTS (

SELECT 1

FROM inserted

WHERE OrderDate IS NULL OR ShipDate IS NULL OR ShipMethod IS NULL

)

BEGIN

RAISERROR(N'Для вставки разрешены только OrderDate, ShipDate и ShipMethod.', 10,10)

END

IF EXISTS (

SELECT 1

FROM inserted

WHERE ShipDate <= OrderDate

)

BEGIN

RAISERROR(N'Дата доставки должна быть больше даты отправки.', 10,10)

END

INSERT INTO AutoPartOrderHeader (OrderDate, ShipDate, ShipMethod)

SELECT

OrderDate,

ShipDate,

ShipMethod

FROM inserted;

END;

GO

1. **Триггер UpdateOrderHeader**

**Описание:** данный триггер вызывается вместо обновления данных в представление AutoPartOrderStatus. Данный триггер обрабатывает данные таким образом, чтобы обновить данные о заказе в отношение AutoPartOrderHeader.

**Подробное описание:**

1. **Триггер срабатывает вместо вставки.** Этот триггер срабатывает вместо операции вставки UPDATE в представление AutoPartOrderStatus.
2. **SET NOCOUNT ON:** устанавливает вариант NOCOUNT для предотвращения возвращения счетчика строк.
3. **Проверка, что обновляются значения только для допустимых атрибутов.**
4. **Проверка правильности обновляемых данных.** Дата доставки должна быть больше даты заказа.

CREATE TRIGGER UpdateOrderHeader

ON AutoPartOrderStatus

INSTEAD OF UPDATE

AS

BEGIN

SET NOCOUNT ON;

IF EXISTS (

SELECT 1

FROM inserted

WHERE OrderDate IS NULL OR ShipDate IS NULL OR ShipMethod IS NULL

)

BEGIN

RAISERROR(N'Для вставки разрешены только OrderDate, ShipDate и ShipMethod.', 10,10)

END

IF EXISTS (

SELECT 1

FROM inserted

WHERE ShipDate <= OrderDate

)

BEGIN

RAISERROR(N'Дата доставки должна быть больше даты отправки.', 10,10)

END

UPDATE oh

SET

oh.OrderDate = i.OrderDate,

oh.ShipDate = i.ShipDate,

oh.ShipMethod = i.ShipMethod

FROM AutoPartOrderHeader oh

INNER JOIN inserted i ON oh.OrderID = i.OrderID;

END;

GO

# **8. Примеры работы с БД с использованием созданных объектов**

Попробуем вставить нового клиента в базу данных. Используем процедуру AddNewCustomer.

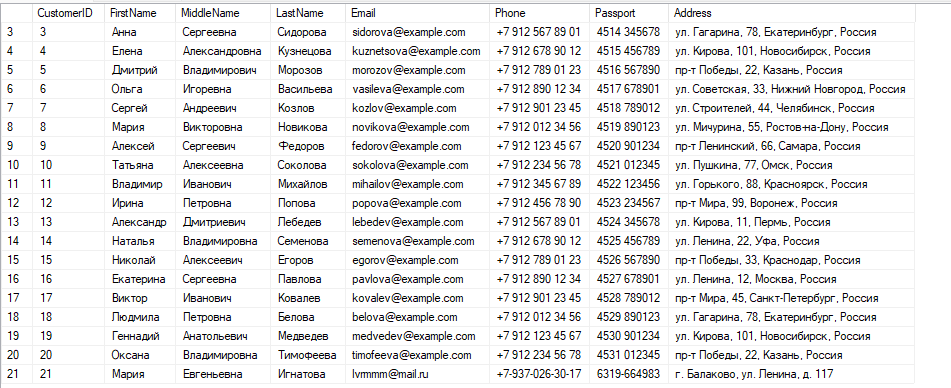


Рис. 2 База данных до вставки нового клиента

EXEC AddNewCustomer @FirstName = 'Павел', @MiddleName = 'Михайлович', @LastName = 'Ленский', @Email='wusuusa@mail.ru',

@Phone = '+7-937-321-53-77', @Passport = '6219-616272', @Address = 'г. Коломна, ул. Карла Маркса, д. 1';

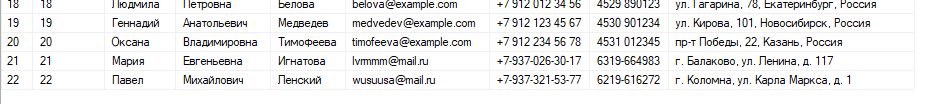


Рис. 3 База данных полсе вставки нового клиента

Попробуем вставить клиента, у которого ФИО и паспорт совпадают с предыдущим клиентом. Очевидно, что паспорт однозначно определяет человека, а значит мы пытаемся вставить уже существующего клиента

EXEC AddNewCustomer @FirstName = 'Павел', @MiddleName = 'Михайлович', @LastName = 'Ленский', @Email='wu112u1sa@mail.ru',

@Phone = '+7-931-321-53-73', @Passport = '6219-616272', @Address = 'г. Бухарест, ул. Карла Маркса, д. 1';

Нам выдало ошибку, значит программа отработала корректно.



Рис. 4 Ошибка при вставке существующего клиента

Теперь мы хотим сравнить характеристики двух машин.

С помощью представления AvailableCarsForSale узнаем, какие машины доступны к покупке.



Рис. 5 Использование представления AvailableCarsForSale

Теперь сравним характеристики машин с CarID1 = 39 и CarID2 = 31. Для этого воспользуемся процедурой CompareCars.

EXEC CompareCars @CarID1 = 39, @CarID2 = 31;

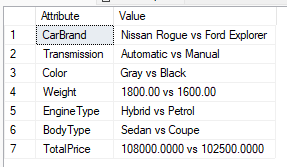


Рис. 6 Результат работы процедуры CompareCars

На основании этих данных клиенту проще определиться с выбором, какую машину покупать.

Допустим, мы хотим сравнить машину с CarID1 = 39 с машиной, которой нет в автосалоне. Например, её CarID2 = 110.

Нам выдало ошибку, программа отработала корректно.



Рис. 7 Ошибка 1 при работе процедуры CompareCars

Попробуем сравнивать машину с самой собой.

EXEC CompareCars @CarID1 = 39, @CarID2 = 39;

Также выдает ошибку, что означает, что программа работает корректно.



Рис. 8 Ошибка 2 при работе процедуры CompareCars

Внесём в Sales новую запись.

Продадим клиенту с ID = 1, машину с ID = 31. ID менеджера, заключающего сделку, 8. Дата продажи сегодняшняя, дата отправки – завтра, дата доставки – через 5 дней с момента продажи.

DECLARE @Date1 DATETIME = GETDATE();

DECLARE @Date2 DATETIME = GETDATE()+1;

DECLARE @Date3 DATETIME = GETDATE()+4;

EXEC AddSale

@CustomerID = 1,

@ManagerID = 8,

@CarID = 31,

@SaleDate = @Date1,

@PaymentStatus = 1,

@DateOfShipment = @Date2,

@DateOfDelivery = @Date3;

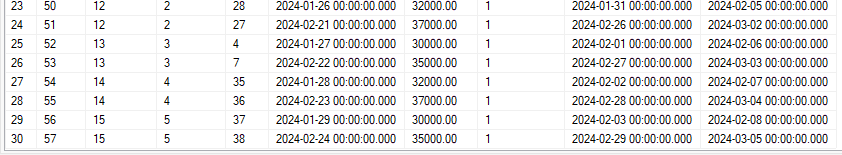


Рис. 9 Данные в Sales до вставки

Программа отработала корректно, изменения внеслись в Sales.

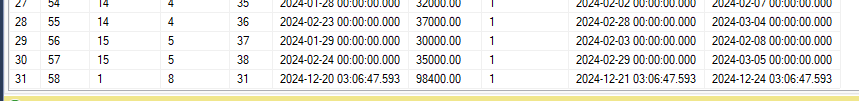


Рис. 10 Данные в Sales после вставки

Проверим значение PurchaseStatus у купленной машины.

Выведено пустое представление, а значит статус машины с CarID = 31 поменялся на 1 и триггер UpdateCarPurchaseStatus отработал корректно.

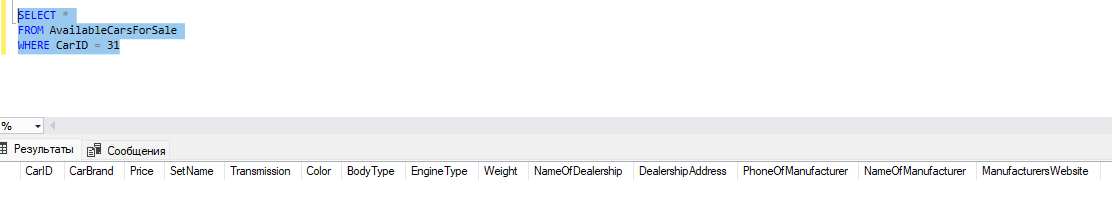


Рис. 11 Проверка возможности покупки машины с CarID = 31

Проверим это также простым SQL запросом.

SELECT PurchaseStatus

FROM Cars

WHERE CarID = 31;

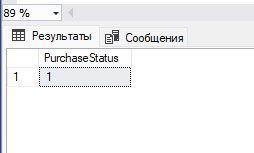


Рис. 12 Результат запроса для проверки статуса покупки машины

Теперь попробуем внести запись с данными, где машина находится в одном филиале, а менеджер закреплен за другим филиалом.

DECLARE @Date1 DATETIME = GETDATE();

DECLARE @Date2 DATETIME = GETDATE()+1;

DECLARE @Date3 DATETIME = GETDATE()+4;

EXEC AddSale

@CustomerID = 5,

@ManagerID = 1,

@CarID = 21,

@SaleDate = @Date1,

@PaymentStatus = 1,

@DateOfShipment = @Date2,

@DateOfDelivery = @Date3;

Нам выдало ошибку, значит программ отработала корректно.

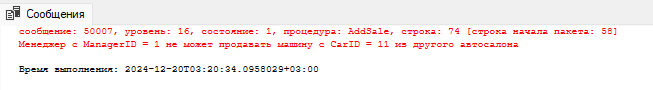


Рис. 13 Ошибка при внесении записи в Sales

Создадим новый заказ на автозапчасти с помощью процедуры CreateAutoPartOrderHeader и укажем его детали с помощью процедуры AddAutoPartOrderDetail.

DECLARE @NewOrderID INT;

EXEC @NewOrderID = CreateAutoPartOrderHeader @OrderDate = '2024-20-12', @ShipDate = '2024-25-12', @ShipMethod = 'Standart';

PRINT 'Создан новый заказ с OrderID: ' + CAST(@NewOrderID AS NVARCHAR);

EXEC AddAutoPartOrderDetail @OrderID = @NewOrderID, @AutoPartID = 10, @UnitPrice = 50.00, @Quantity = 2;

EXEC AddAutoPartOrderDetail @OrderID = @NewOrderID, @AutoPartID = 11, @UnitPrice = 50.00, @Quantity = 2;

EXEC AddAutoPartOrderDetail @OrderID = @NewOrderID, @AutoPartID = 4, @UnitPrice = 1000.0, @Quantity = 5;

Изменения были внесены, программа отработала корректно.

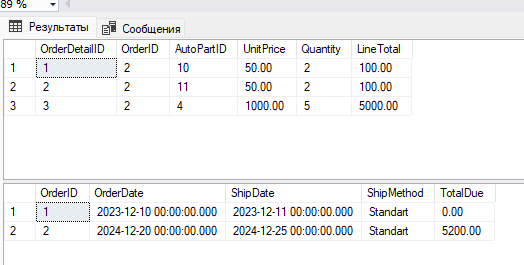


Рис. 14 Работа процедур CreateAutoPartOrderHeader и AddAutoPartOrderDetail

Можем заметить, что при работе программы автоматически посчитались значения LineTotal и TotalDue, а значит триггеры CalculateLineTotal и CalculateTotalDue отработали корректно.

Для отслеживания статуса заказа на автозапчасти используем представление AutoPartOrderStatus.

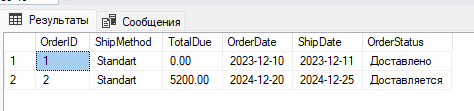


Рис. 15 Данные из представления AutoPartOrderStatus

Поменяем дату доставки заказа 2 на 26 декабря 2024 года.

UPDATE AutoPartOrderStatus

SET ShipDate = '2024-12-26'

WHERE OrderID = 2;

Снова извлечём все данные из представления AutoPartOrderStatus.

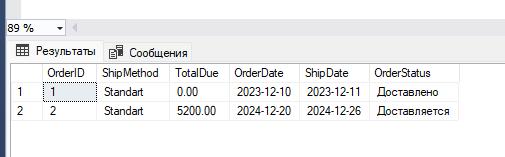


Рис. 16 Данные из представления AutoPartOrderStatus после обновления

Видим, что данные о дате доставки в заказе 2 изменились, а значит триггер UpdateOrderHeader сработал корректно.

# **Список литературы**

1. Документация Microsoft SQL Документация по Microsoft SQL- SQL Server | Microsoft Learn
2. Курс лекций по предмету «Базы данных» Валовой А.А.