Министерство образования и науки

Санкт-Петербургский национальный исследовательский университет информационных технологий, механики и оптики

Факультет инфокоммуникационных технологий

Отчет

по лабораторным работам

по дисциплине **«Базы данных»**

Выполнил: **студент группы K3220 И.О. Мороз**

Проверил: **ст. преподаватель И.С. Осетрова**

Санкт-Петербург

2020

# «Создание учебной базы данных»

## Цель работы.

Создание учебной базы данных с использованием SQL запросов. Создание сценария базы данных.

## Задачи, решаемые при выполнении работы.

### Подключение к Microsoft SQL Server.

### Создание учебной базы данных.

### Создание сценария базы данных.

### Удаление учебной базы данных.

### Создание учебной базы данных в Query Editor.

## Объект исследования.

Microsoft SQL Server, учебная база данных.

## Исходные данные.

### Docker-контейнер с SQL Server, DataGrip IDE.

### Методическое пособие.

### Методическое пособие.

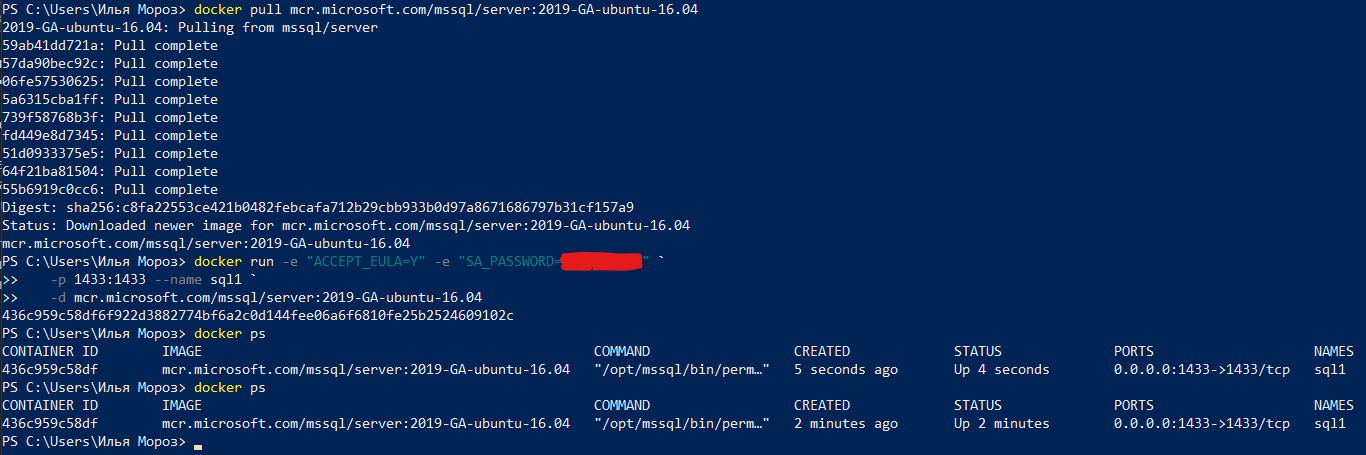
### Методическое пособие.

### Методическое пособие.

## Выполнение работы.

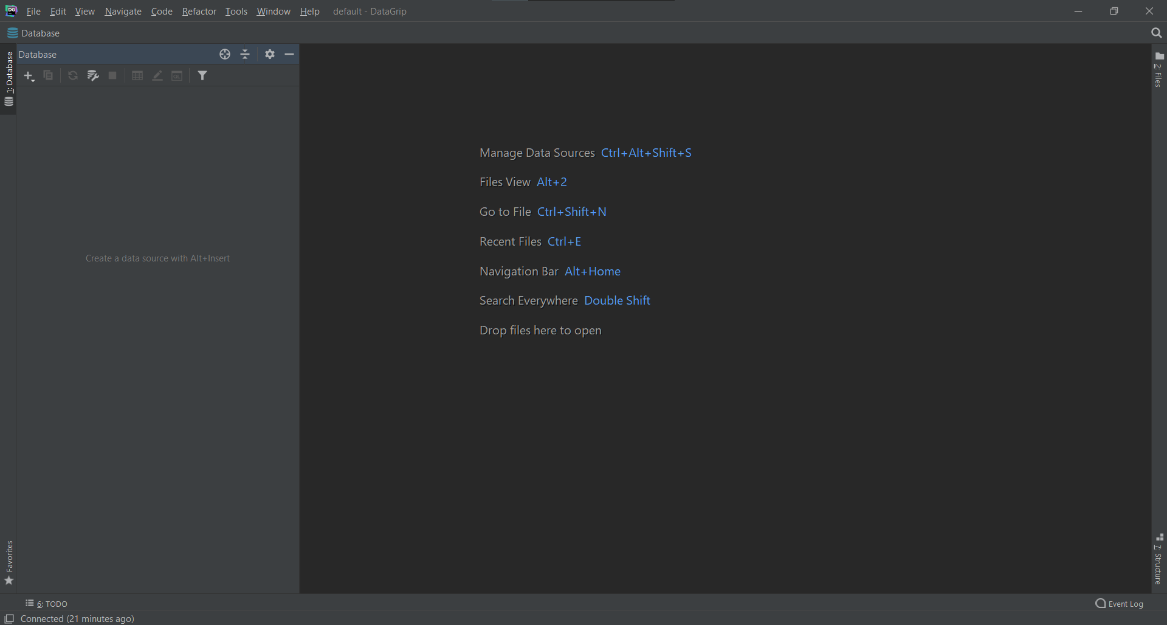
### Подключение к Microsoft SQL Server.

Перед выполнением задания было необходимо поднять SQL Server в локальной сети и подключиться к ней. Для упрощения процесса поднятия SQL Server был использован Docker контейнер от Microsoft (подробнее см. [Краткое руководство. Запуск образов контейнеров SQL Server в Docker](https://docs.microsoft.com/ru-ru/sql/linux/quickstart-install-connect-docker?view=sql-server-ver15&pivots=cs1-cmd))

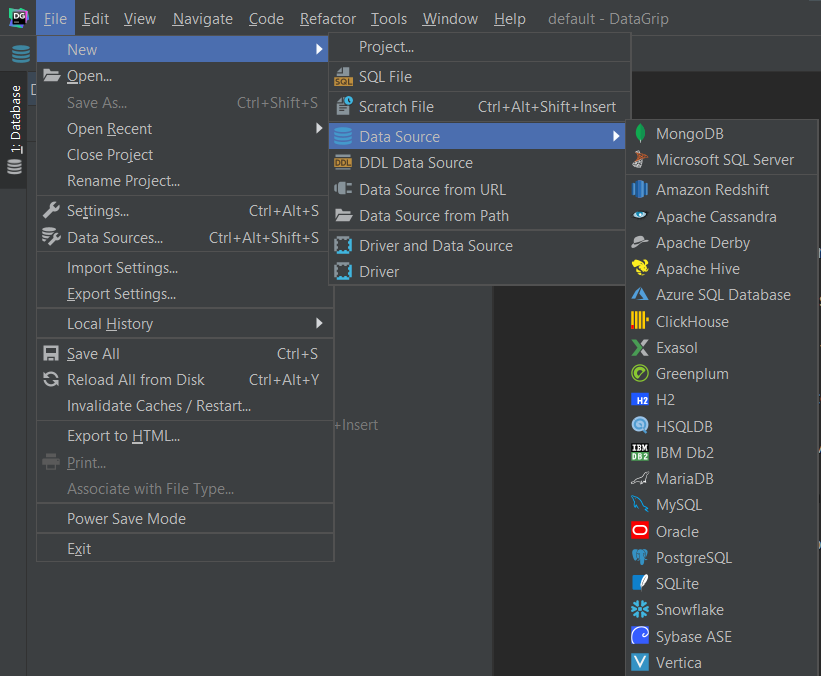


*Рисунок 1 – Запуск Docker контейнера с SQL Server*

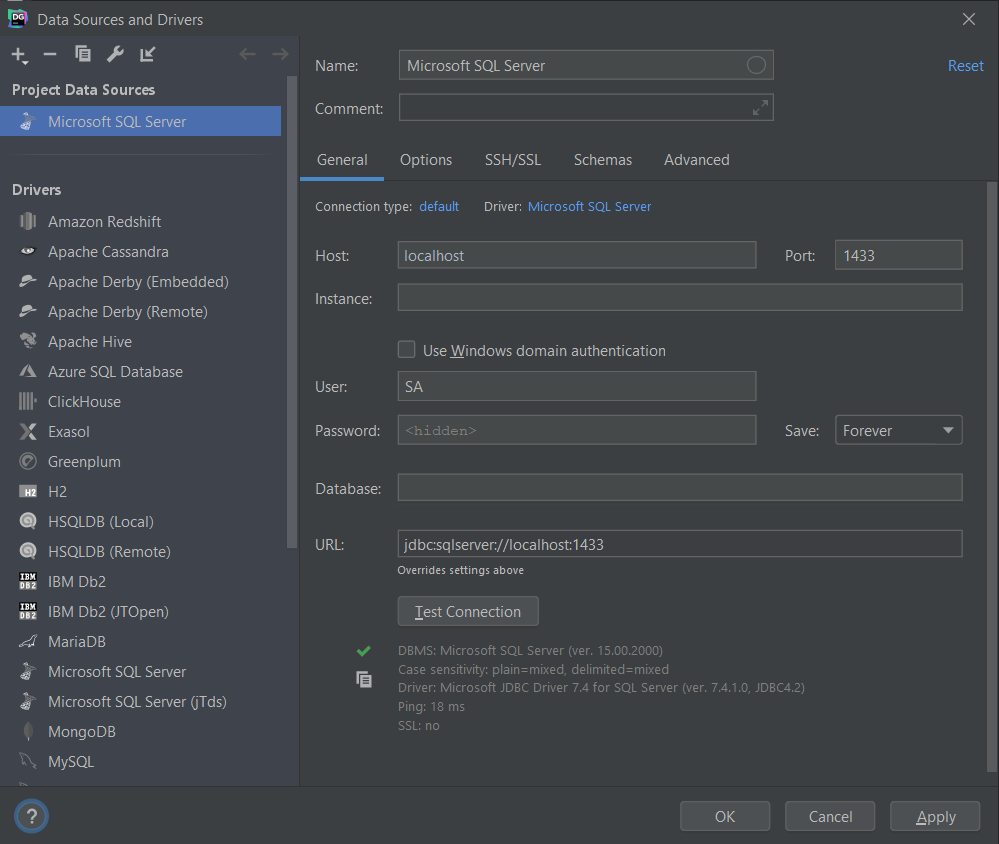
После запуска SQL Server подключение было выполнено с использованием DataGrip:



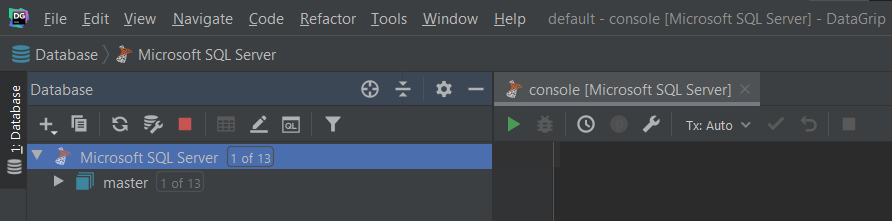
*Рисунок 2 – Окно DataGrip IDE*



*Рисунок 3 – Создание нового Data Source. File – New – Data Source – Microsoft SQL Server*



*Рисунок 4 – Настройка нового подключения*



*Рисунок 5 – Успешное подключение к SQL Server*

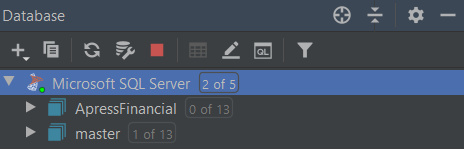
### Создание учебной базы данных.

С помощью SQL запроса была создана новая база ApressFinancial. При создании базы данных был создан вторичный файл данных с логическим именем «ApressFinancial\_act»

CREATE DATABASE ApressFinancial  
ON  
 PRIMARY  
 ( NAME = N'ApressFinancial',  
 FILENAME = N'/var/opt/mssql/data/ApressFinancial.mdf',  
 SIZE = 8192KB,  
 MAXSIZE = UNLIMITED,  
 FILEGROWTH = 1024KB ),  
 FILEGROUP [SECONDARY] DEFAULT  
 ( NAME = N'ApressFinancial\_act',  
 FILENAME = N'/var/opt/mssql/data/ApressFinancial\_act.ndf',  
 SIZE = 8192KB,  
 MAXSIZE = UNLIMITED,  
 FILEGROWTH = 1024KB )  
LOG ON  
 ( NAME = N'ApressFinancial\_log',  
 FILENAME = N'/var/opt/mssql/data/ApressFinancial\_log.ldf',  
 SIZE = 1024KB,  
 MAXSIZE = 2048KB,  
 FILEGROWTH = 10% )  
GO

*Листинг 1 – SQL запрос для создания базы данных «ApressFinancial»*

После выполнения данного SQL запроса была создана база данных.



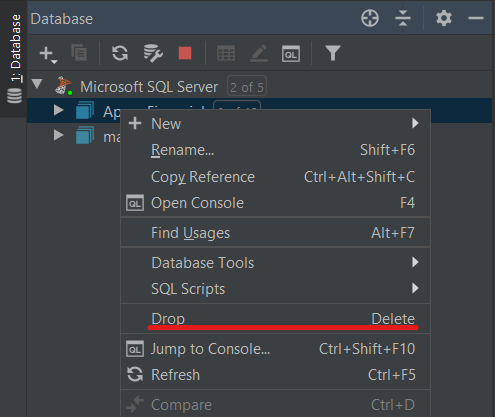
*Рисунок 6 – База данных создана*

### Создание сценария базы данных.

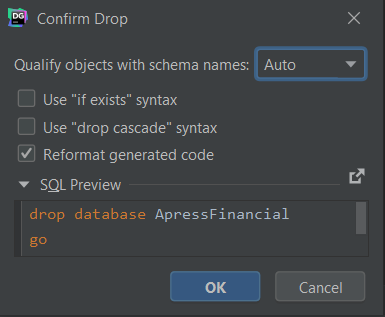
Так как DataGrip IDE поддерживает создание базы данных только с помощью SQL запроса, в предыдущем задании был создан файл сценария базы данных *create\_database.sql*. Листинг этого файла прикреплён в задании 1 (см. Листинг 1).

### Удаление учебной базы данных.

В данном задании было необходимо удалить только что созданную базу данных ApressFinancial.

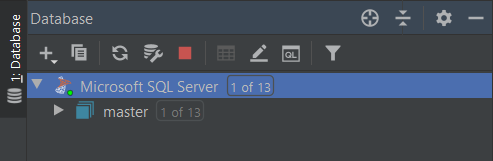


*Рисунок 7 – Команда для удаления базы данных в контекстном меню*



*Рисунок 8 – Подтверждение удаления базы данных*

База данных успешно удалена.

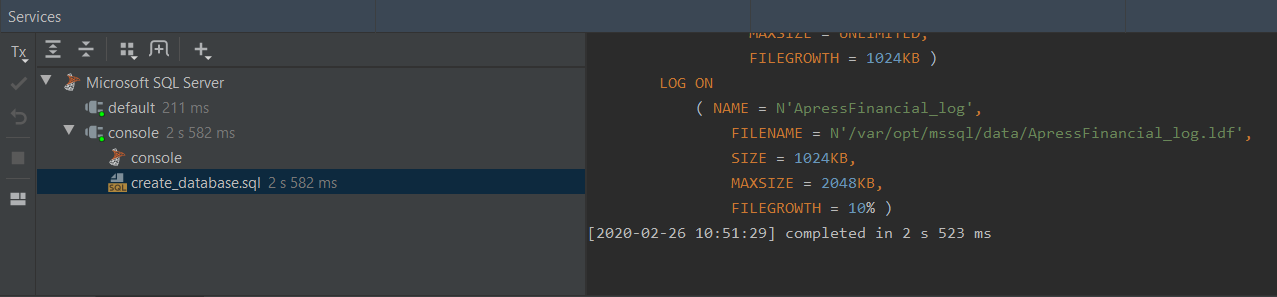


*Рисунок 9 – Структура Microsoft SQL Server после удаления базы данных*

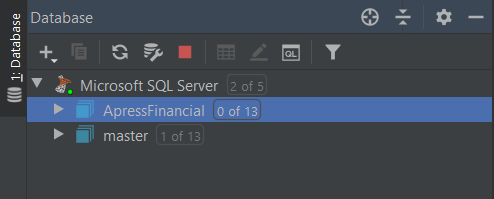
### Создание учебной базы данных в Query Editor.

Для создания учебной базы данных с помощью SQL запроса был открыт уже созданный файл *create\_database.sql*.

Файл был выполнен.



*Рисунок 10 – Результат выполнения файла*

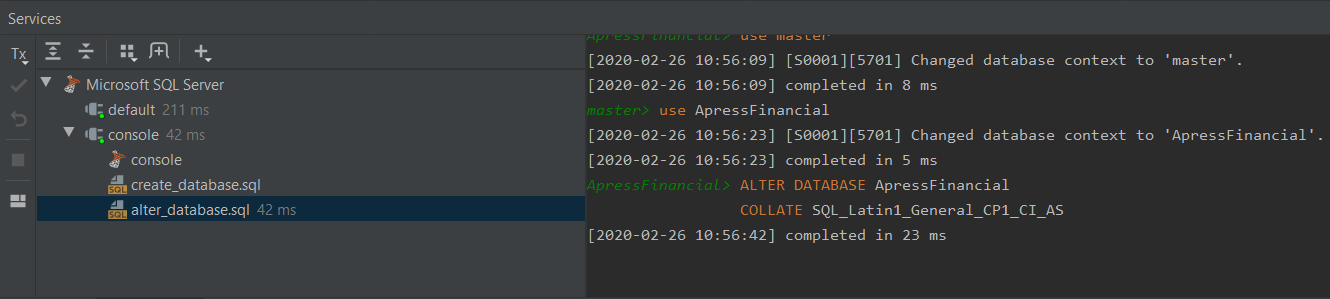


*Рисунок 11 – База данных вновь создана*

Для изменения параметров сортировки для символьных строк был создан файл SQL инструкций *alter\_database.sql*.

ALTER DATABASE ApressFinancial  
COLLATE SQL\_Latin1\_General\_CP1\_CI\_AS  
GO

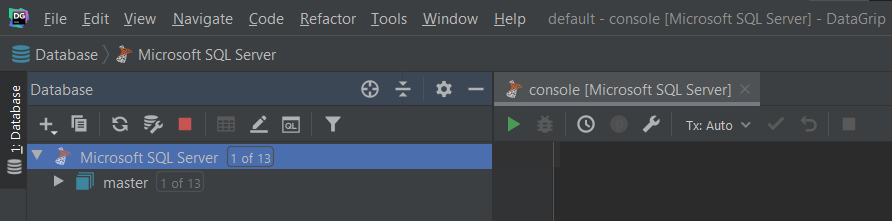
*Листинг 2 – Изменение параметров сортировки для символьных строк*



*Рисунок 12 – Результат выполнения файла*

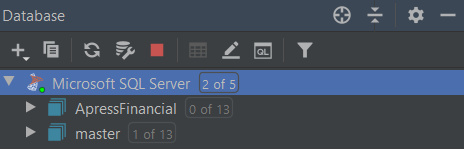
## Окончательные результаты.

### Подключение к базе данных было выполнено.



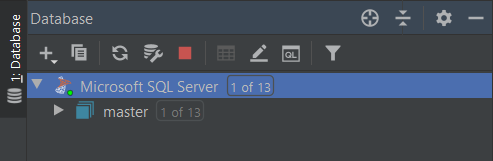
*Рисунок 13 – Успешное подключение к SQL Server*

### Была создана учебная база данных.



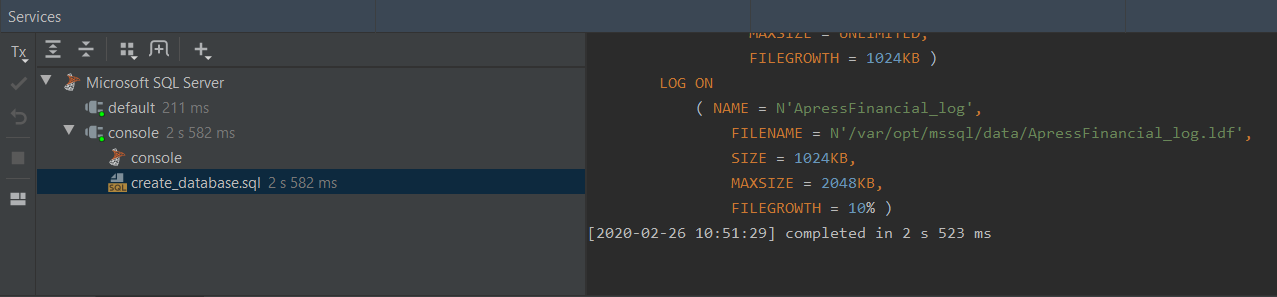
*Рисунок 14 – База данных создана*

### Учебная база данных была удалена с использованием контекстного меню.

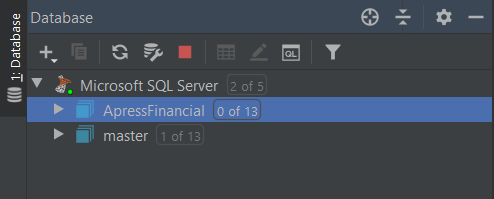


*Рисунок 15 – Структура Microsoft SQL Server после удаления базы данных*

### С помощью Query Editor был создан новый экземпляр учебной базы данных.



*Рисунок 16 – Результат выполнения файла*



*Рисунок 17 – База данных вновь создана*

## Выводы и анализ результатов работы.

В ходе данной лабораторной работы локально был запущен Microsoft SQL Server. При запуске использовался Docker контейнер. К созданному SQL Server было выполнено подключение с использованием DataGrip IDE.

При выполнении заданий были созданы файлы SQL сценариев *create\_database.sql* и *alter\_database.sql*. При выполнении этих файлов была создана учебная база данных ApressFinancial. Для данной базы данных были изменены параметры сортировки для символьных строк.

# «Безопасность: создание схем»

## Цель работы.

Создание схем для базы данных с использованием контекстного меню и Query Editor.

## Задачи, решаемые при выполнении работы.

### Создание новой схемы с использованием контекстного меню.

### Создание новой схемы с использованием Query Editor.

## Объект исследования.

Microsoft SQL Server, учебная база данных, схемы базы данных.

## Исходные данные.

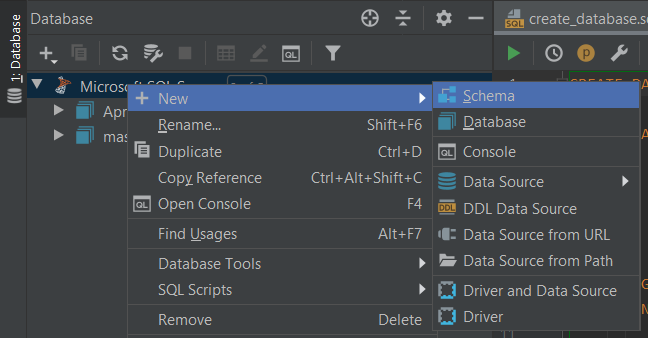
### Методическое пособие, учебная база данных.

### Методическое пособие, учебная база данных.

## Выполнение работы.

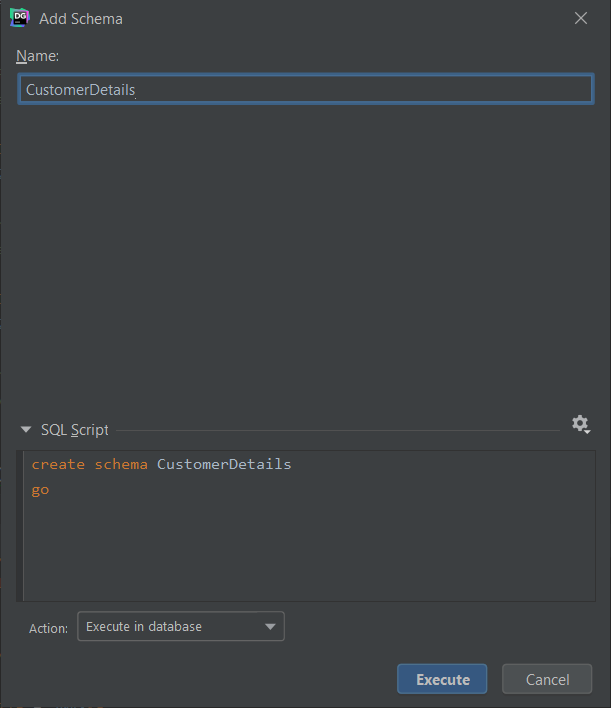
### Создание схемы в DataGrip IDE.

Для создания новой схемы необходимо нажать ПКМ на Data source, выбрать New – Schema.



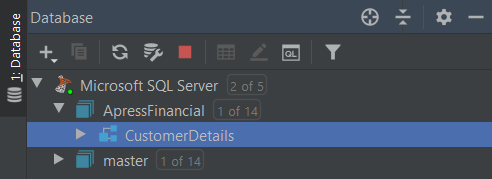
*Рисунок 18 – Контекстное меню создания новой схемы*

В появившемся окне было введено имя схемы «CustomerDetails».



*Рисунок 19 – Окно создания новой схемы*

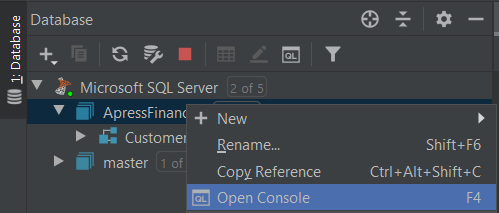
После создания новой схемы она отобразилась у базы данных ApressFinancial.



*Рисунок 20 – Только что созданная схема CustomerDetails*

### Создание схемы с помощью SQL запроса.

Для создания схемы с помощью SQL запроса необходимо открыть консоль для БД ApressFinancial.



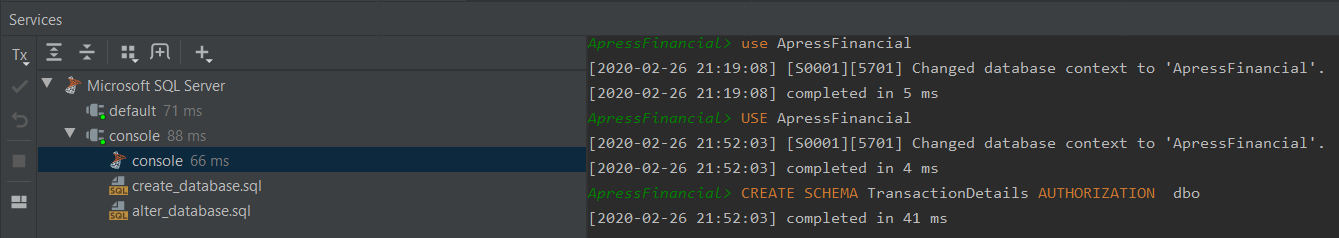
*Рисунок 21 – Открытие консоли*

В консоли был введён следующий SQL запрос.

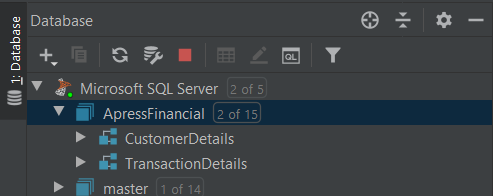
USE ApressFinancial  
GO  
  
CREATE SCHEMA TransactionDetails AUTHORIZATION dbo  
GO

*Листинг 3 – Создание схемы TransactionDetails*

Данный запрос был успешно выполнен.



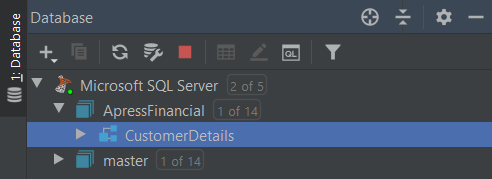
*Рисунок 22 – Успешное выполнение SQL запроса*



*Рисунок 23 – Схема появилась у базы данных*

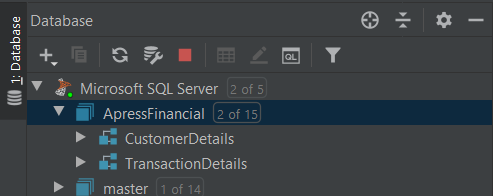
## Окончательные результаты.

### С помощью контекстного меню схема была создана.



*Рисунок 24 – Только что созданная схема CustomerDetails*

### С помощью SQL запроса схема была создана.



*Рисунок 25 – Схема появилась у базы данных*

## Выводы и анализ результатов работы.

В ходе данной лабораторной работы локально были созданы схемы базы данных с помощью контекстного меню DataGrip IDE и с помощью SQL запроса. В результате был получен файл с запросом SQL *create\_schema.sql*.

# «Проектирование и создание Таблиц»

## Цель работы.

Создание и проектирование таблиц в учебной базе данных с помощью контекстного меню и SQL запросов.

## Задачи, решаемые при выполнении работы.

### Создание таблицы с использованием контекстного меню.

### Создание таблицы в Query Editor.

### Создание таблицы с помощью шаблона.

### Изменение таблицы.

### Создание связи с помощью контекстного меню.

### Создание связи с помощью SQL запроса.

## Объект исследования.

Microsoft SQL Server, учебная база данных, таблицы базы данных.

## Исходные данные.

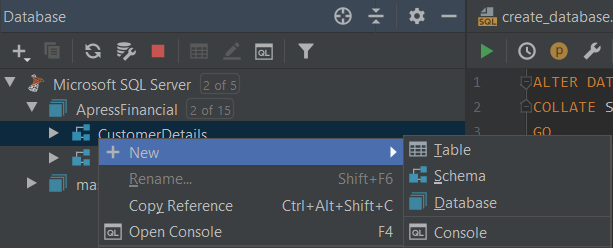
### Методическое пособие.

### Учебная база данных.

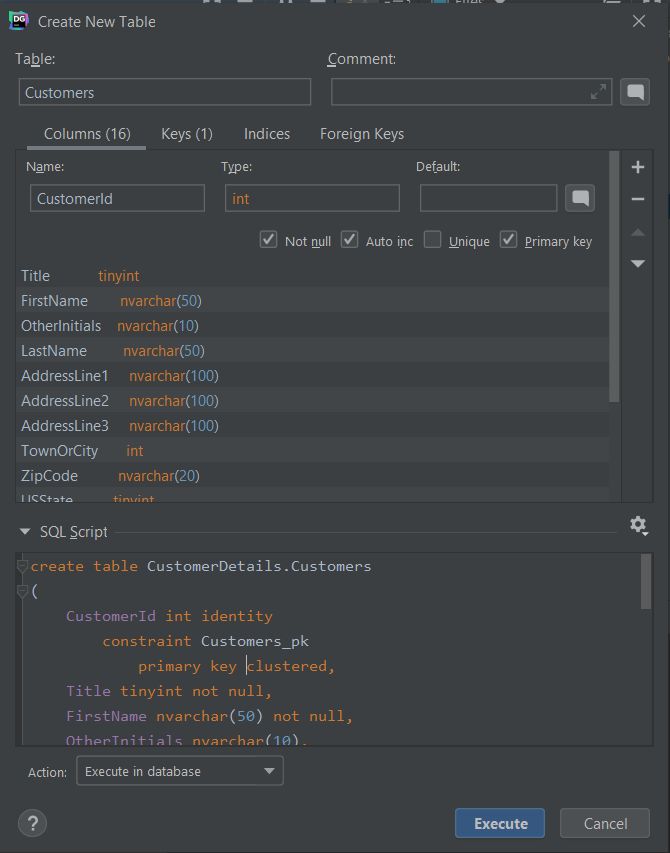
## Выполнение работы.

### Создание таблицы с использованием контекстного меню в DataGrip IDE.

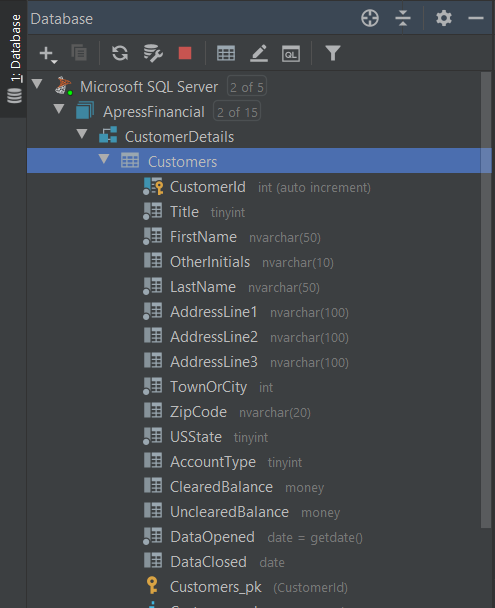
Для создания таблицы в DataGrip IDE необходимо создать схему. Так как создаётся таблица *Customers*, была использована уже существующая схема *CustomerDetails*.



*Рисунок 26 – CustomerDetails – ПКМ – New – Table*



*Рисунок 27 – Окно создания новой таблицы Customers*

**

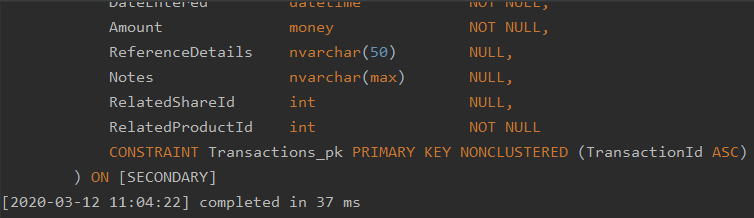
*Рисунок 28 – Таблица успешно создана*

### Создание таблицы с помощью Query Editor

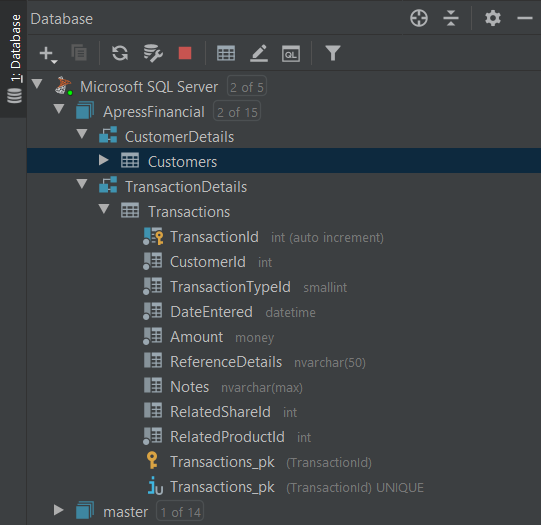
Был создан файл *create\_transactions\_table.sql* со следующим содержимым.

USE ApressFinancial  
GO  
  
CREATE TABLE TransactionDetails.Transactions  
(  
 TransactionId int IDENTITY(1,1) NOT NULL,  
 CustomerId int NOT NULL,  
 TransactionTypeId smallint NOT NULL,  
 DateEntered datetime NOT NULL,  
 Amount money NOT NULL,  
 ReferenceDetails nvarchar(50) NULL,  
 Notes nvarchar(max) NULL,  
 RelatedShareId int NULL,  
 RelatedProductId int NOT NULL  
 CONSTRAINT Transactions\_pk PRIMARY KEY NONCLUSTERED (TransactionId ASC)  
) ON [SECONDARY]  
GO

*Листинг 4 – Запрос для создания таблицы Transactions*

**

*Рисунок 29 – Результат выполнения файла в консоли*

**

*Рисунок 30 – Таблица успешно создана*

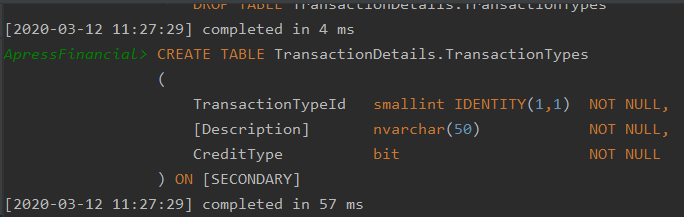
### Создание таблицы *TransactionTypes*

Так как DataGrip IDE не предусматривает создание таблиц с использованием Templates, данная таблица была создана с помощью SQL запроса.

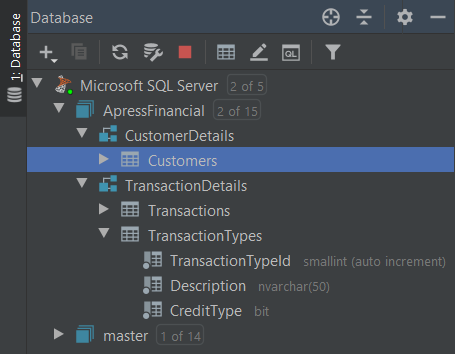
Был создан файл *create\_transactiontypes\_table.sql* со следующим содержимым.

USE ApressFinancial  
GO  
  
IF *OBJECT\_ID*('TransactionDetails.TransactionTypes', 'U') IS NOT NULL  
 DROP TABLE TransactionDetails.TransactionTypes  
GO  
  
CREATE TABLE TransactionDetails.TransactionTypes  
(  
 TransactionTypeId smallint IDENTITY(1,1) NOT NULL,  
 [Description] nvarchar(50) NOT NULL,  
 CreditType bit NOT NULL  
) ON [SECONDARY]  
GO

*Листинг 5 – Код создания таблицы TransactionTypes*



*Рисунок 31 – Результат выполнения кода в консоли*

**

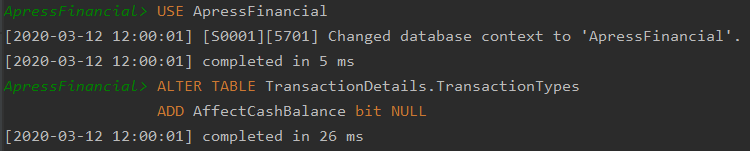
*Рисунок 32 – Таблица успешно создана*

### Изменение таблицы.

Необходимо добавить поле *AffectCashBalance* в уже существующую таблицу *TransactionTypes*. Для этого был создан файл со следующим SQL запросом.

USE ApressFinancial  
GO  
  
ALTER TABLE TransactionDetails.TransactionTypes  
ADD AffectCashBalance bit NULL  
GO

*Листинг 6 – SQL запрос создания поля AffectCashBalance*



*Рисунок 33 – Результат выполнения запроса в консоли. Поле успешно добавлено.*

Поле *AffectCashBalance* было изменено. Теперь оно обязательно должно содержать значения 0 или 1.

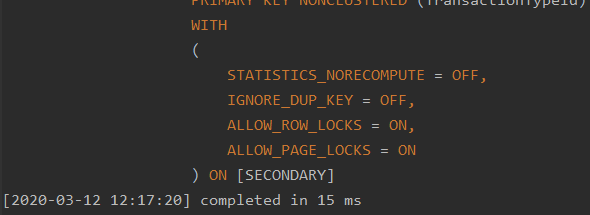
ALTER TABLE TransactionDetails.TransactionTypes  
ALTER COLUMN AffectCashBalance bit NOT NULL

*Листинг 7 – SQL запрос изменения поля AffectCashBalance*

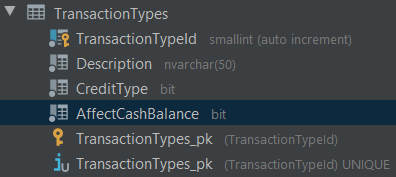
Необходимо создать Primary Key для таблицы TransactionTypes. Был создан файл со следующим содержимым.

USE ApressFinancial  
GO  
  
ALTER TABLE TransactionDetails.TransactionTypes  
ADD CONSTRAINT TransactionTypes\_pk  
 PRIMARY KEY NONCLUSTERED (TransactionTypeId)  
 WITH  
 (  
 STATISTICS\_NORECOMPUTE = OFF,  
 IGNORE\_DUP\_KEY = OFF,  
 ALLOW\_ROW\_LOCKS = ON,  
 ALLOW\_PAGE\_LOCKS = ON  
 ) ON [SECONDARY]  
GO

*Листинг 8 – SQL запрос для создания Primary Key*



*Рисунок 34 – Результат выполнения кода в консоли*



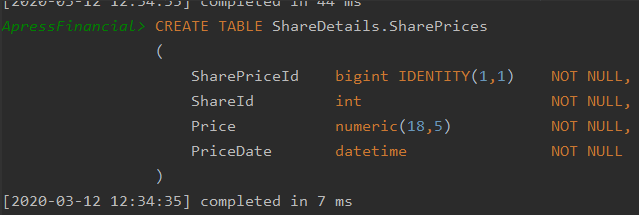
*Рисунок 35 – Таблица TransactionTypes после всех изменений*

### Создание остальных таблиц.

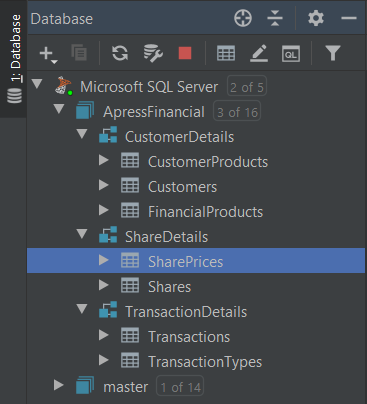
На данном шаге были созданы оставшиеся таблицы с использованием SQL запроса.

USE ApressFinancial  
GO  
  
CREATE TABLE CustomerDetails.FinancialProducts  
(  
 ProductId int NOT NULL,  
 ProductName nvarchar(50) NOT NULL  
) ON [SECONDARY]  
GO  
  
CREATE TABLE CustomerDetails.CustomerProducts  
(  
 CustomerFinancialProductId int NOT NULL,  
 CustomerId int NOT NULL,  
 FinancialProductId int NOT NULL,  
 AmountToCollect money NOT NULL,  
 Frequency smallint NOT NULL,  
 LastCollected datetime NOT NULL,  
 LastCollection datetime NOT NULL,  
 Renewable bit NOT NULL  
) ON [SECONDARY]  
GO  
  
CREATE SCHEMA ShareDetails AUTHORIZATION dbo  
 CREATE TABLE Shares  
 (  
 ShareId int IDENTITY(1,1) NOT NULL,  
 [Description] nvarchar(50) NOT NULL,  
 StockExchangeTicker nvarchar(50) NULL,  
 CurrentPrice numeric(18,5) NOT NULL  
 ) ON [SECONDARY]  
GO  
  
CREATE TABLE ShareDetails.SharePrices  
(  
 SharePriceId bigint IDENTITY(1,1) NOT NULL,  
 ShareId int NOT NULL,  
 Price numeric(18,5) NOT NULL,  
 PriceDate datetime NOT NULL  
)  
GO

*Листинг 9 – SQL запрос для создания оставшихся таблиц*

**

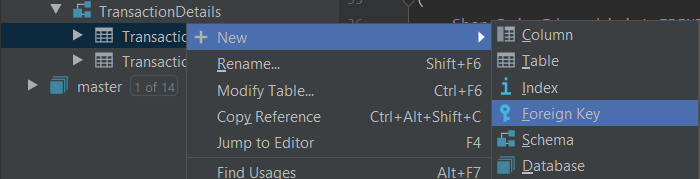
*Рисунок 36 – Результат выполнения кода в консоли*

**

*Рисунок 37 – Учебная база данных после создания новых таблиц*

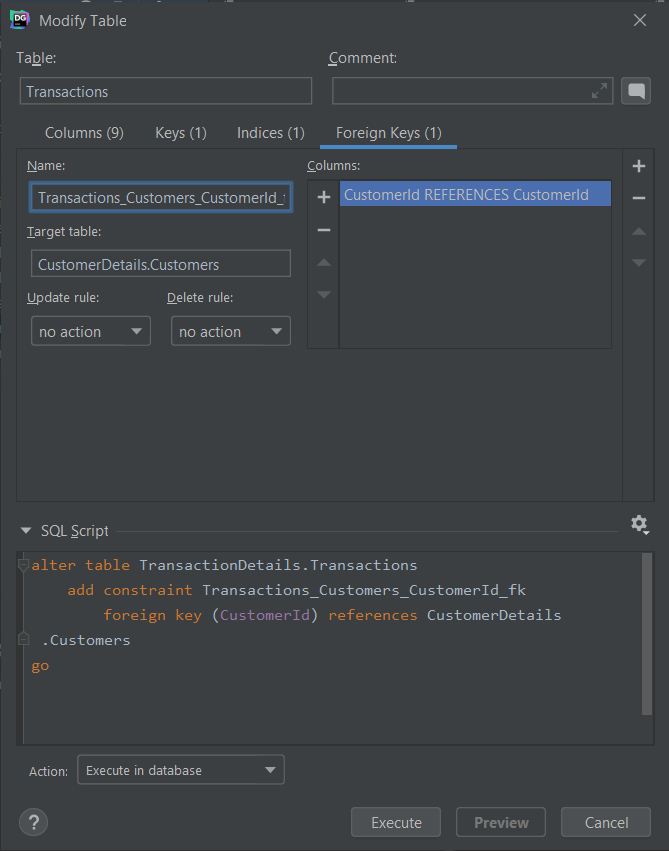
### Создание связи с помощью контекстного меню в DataGrip IDE.

Необходимо создать связь между таблицами *Transactions* и *Customers*.



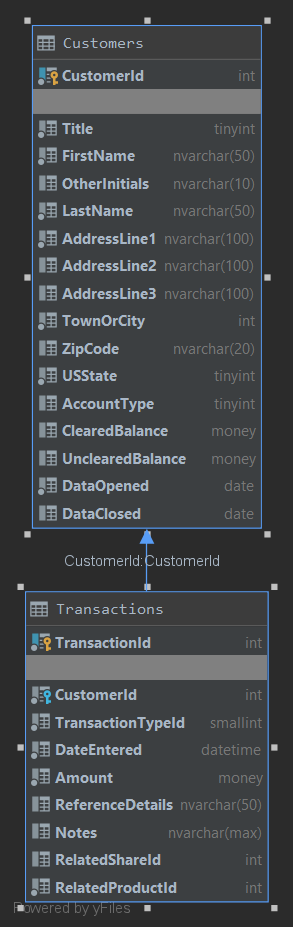
*Рисунок 38 – Transactions – ПКМ – New – Foreign Key*

Были заполнены данные о связи.



*Рисунок 39 – Окно создания новой связи*

После создания связи была создана диаграмма БД, на которой можно увидеть созданную связь.



*Рисунок 40 – Диаграмма базы данных*

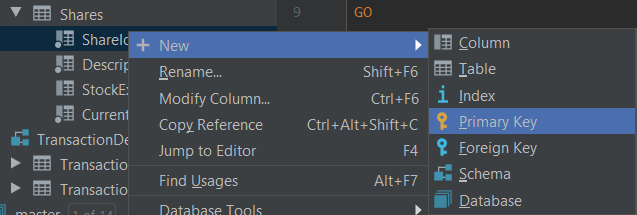
### Создание связи с помощью SQL запроса.

Была создана связь между таблицами *Transactions* и *Shares*. Файл *create\_transactions\_shares\_fk.sql* содержит SQL запрос для создания данной связи.

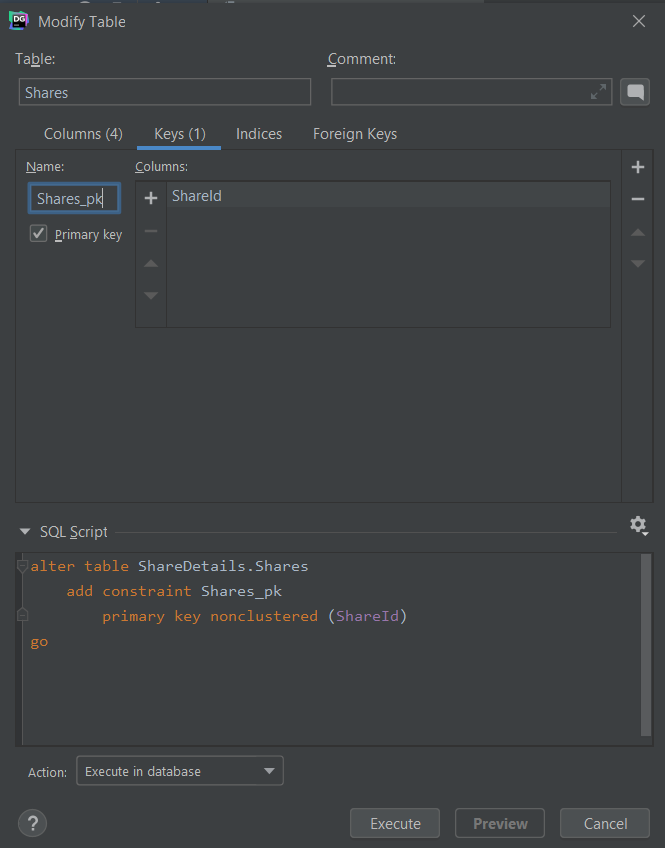
USE ApressFinancial  
GO  
  
ALTER TABLE TransactionDetails.Transactions  
WITH NOCHECK  
ADD CONSTRAINT Transactions\_Shares\_RelatedShareId\_fk  
 FOREIGN KEY (RelatedShareId)  
 REFERENCES ShareDetails.Shares(ShareId)  
GO

*Листинг 10 – Код создания связи*

Данный код при выполнении выдаёт ошибку. Это связано с тем, что в таблице *Shares* поле *ShareId* не является *Primary Key*.

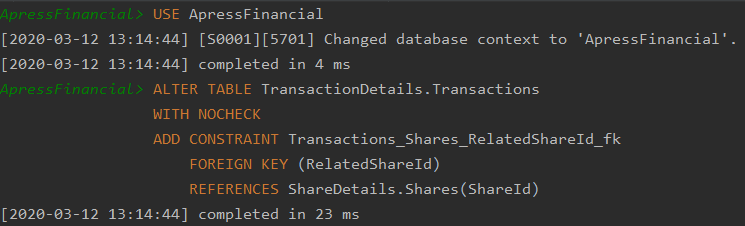


*Рисунок 41 – ShareId – ПКМ – New – Primary Key*

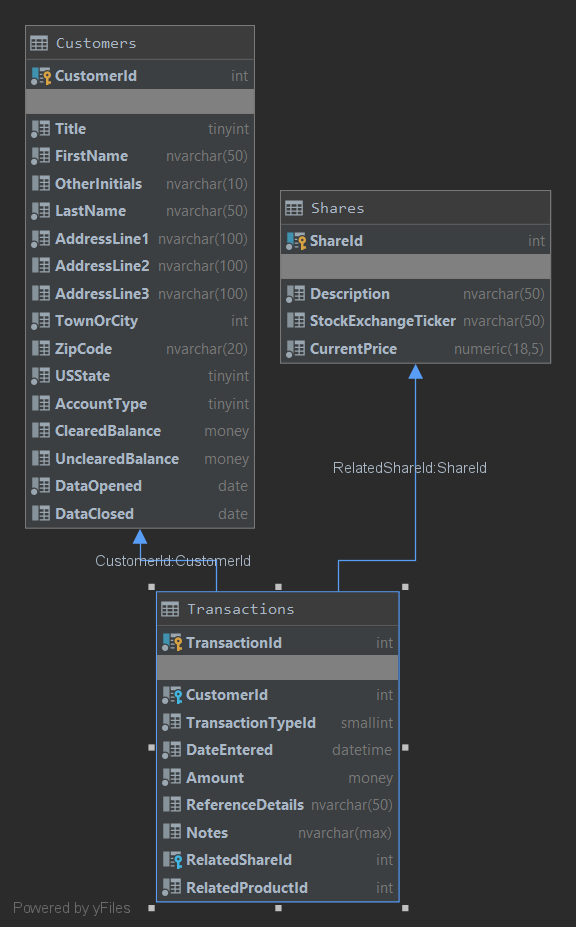


*Рисунок 42 – Окно создания Primary Key*

Теперь код успешно выполнился.



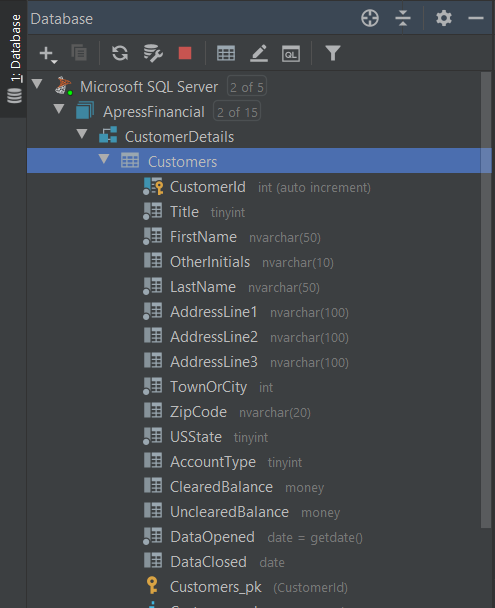
*Рисунок 43 – Результат выполнения кода в консоли*



*Рисунок 44 – новая схема учебной базы данных*

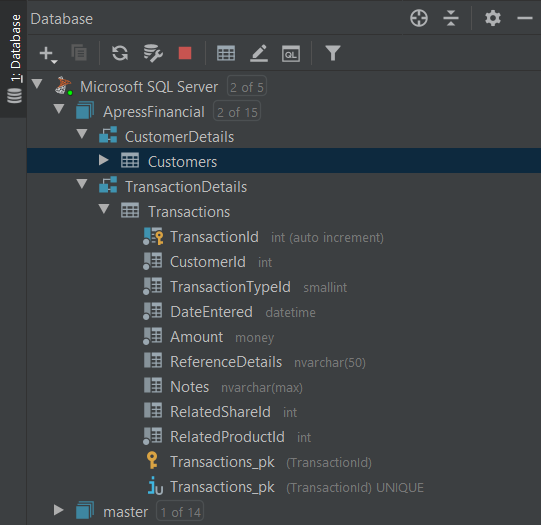
## Окончательные результаты.

### С помощью контекстного меню таблица была успешно создана.

**

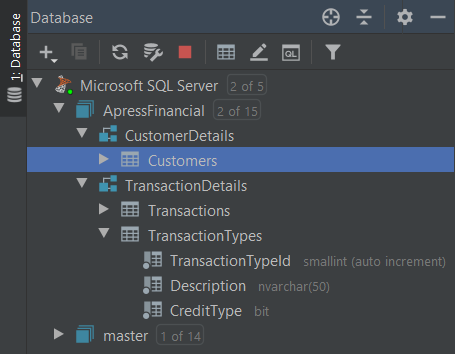
*Рисунок 45 – Таблица успешно создана*

### С помощью SQL запроса таблица была успешно создана.

**

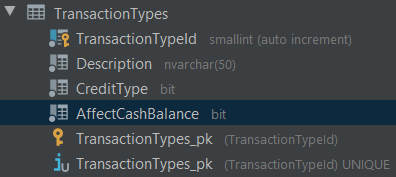
*Рисунок 46 – Таблица успешно создана*

### С помощью SQL запроса была успешно создана новая таблица.

**

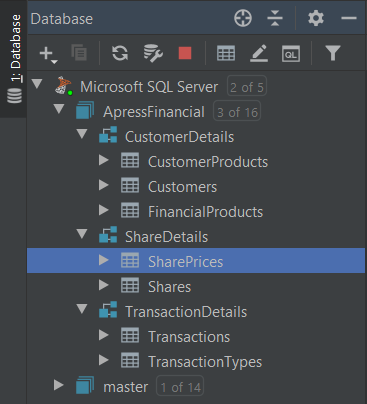
*Рисунок 47 – Таблица успешно создана*

### С помощью SQL запроса таблица *TransactionTypes* была изменена.



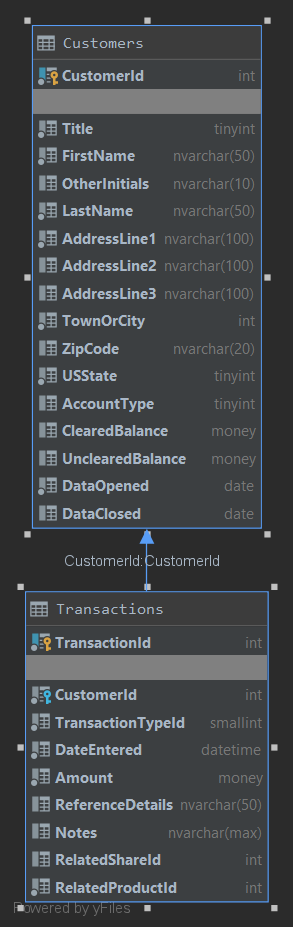
*Рисунок 48 – Таблица TransactionTypes после всех изменений*

### Были созданы оставшиеся таблицы.

**

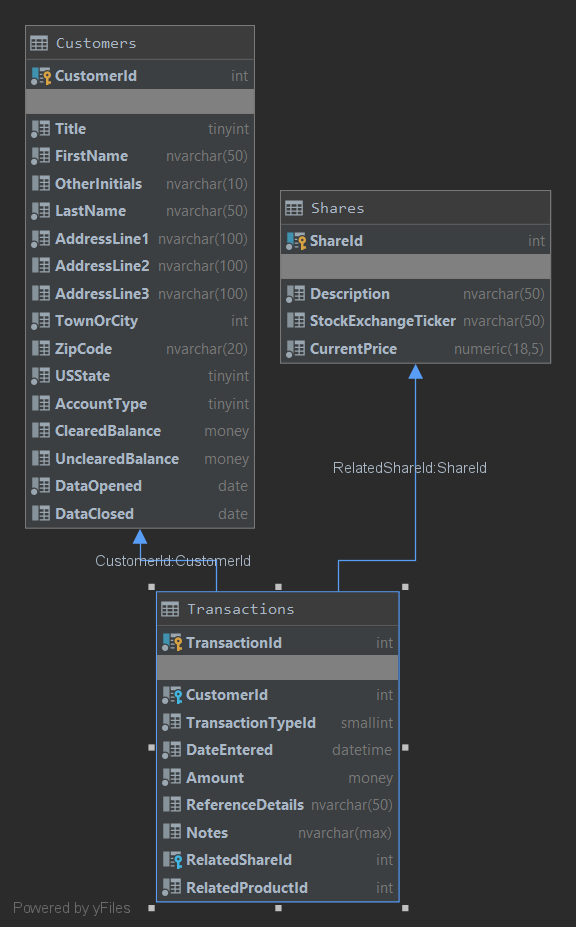
*Рисунок 49 – Учебная база данных после создания новых таблиц*

### С помощью контекстного меню была создана новая связь.



*Рисунок 50 – Диаграмма базы данных*

### С помощью SQL запроса была создана новая связь.



*Рисунок 51 – новая схема учебной базы данных*

## Выводы и анализ результатов работы.

В ходе данной лабораторной работы локально были созданы таблицы базы данных с помощью контекстного меню DataGrip IDE и с помощью SQL запроса. Были созданы связи между таблицами.

# «Создание Индексов и Диаграмм»

## Цель работы.

Создание индексов и построение диаграмм в учебной базе данных с помощью контекстного меню и SQL запросов.

## Задачи, решаемые при выполнении работы.

### Создание индекса с использованием контекстного меню конструктора таблиц.

### Создание индекса в Query Editor.

### Изменение индекса.

### Удаление индекса.

### Построение диаграмм базы данных.

## Объект исследования.

Microsoft SQL Server, учебная база данных, индексы и диаграммы базы данных.

## Исходные данные.

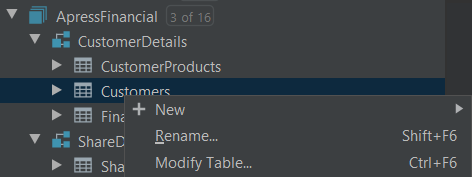
### Методическое пособие.

### Учебная база данных.

## Выполнение работы.

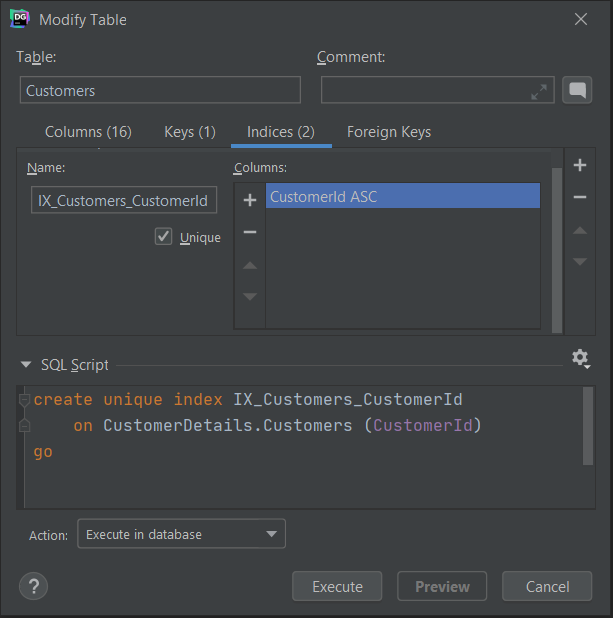
### Создание индекса с помощью конструктора таблиц.

Создадим новый индекс для таблицы Customers.



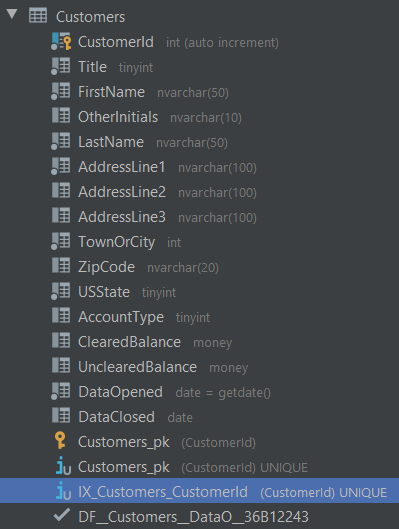
*Рисунок 52 – Редактирование таблицы: Customers – ПКМ – Modify Table…*

В окне Modify Table переходим во вкладку Indices.



*Рисунок 53 – Окно создания нового индекса*

Был создан новый индекс *IX\_Customers\_CustomerId*.



*Рисунок 54 – Новый индекс был добавлен в таблицу Customers*

### Создание индекса с использованием Query Editor.

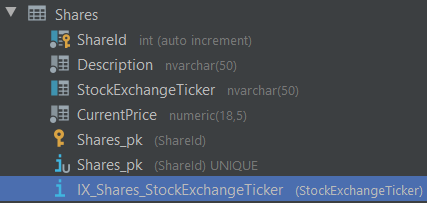
Было отработано создание индексов с использованием Query Editor.

Был создан новый индекс *IX\_Shares\_StockExchangeTicker* с использованием команд T-SQL.

USE ApressFinancial  
GO  
  
CREATE INDEX IX\_Shares\_StockExchangeTicker  
ON ShareDetails.Shares (StockExchangeTicker)  
GO

*Листинг 11 – Код создания индекса IX\_Shares\_StockExchangeTicker*

В результате выполнения кода был создан новый индекс.



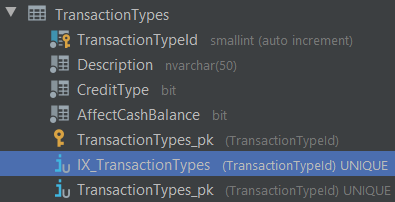
*Рисунок 55 – Создан новый индекс в таблице Shares*

Был создан новый индекс *IX\_TransactionTypes* с использованием команд T-SQL.

USE ApressFinancial  
GO  
  
CREATE UNIQUE CLUSTERED INDEX IX\_TransactionTypes  
ON TransactionDetails.TransactionTypes (TransactionTypeId ASC)  
WITH ( STATISTICS\_NORECOMPUTE = ON, SORT\_IN\_TEMPDB = ON,  
 DROP\_EXISTING = OFF, IGNORE\_DUP\_KEY = OFF, ONLINE = ON,  
 ALLOW\_ROW\_LOCKS = ON, ALLOW\_PAGE\_LOCKS = ON ) ON [SECONDARY]  
GO

*Листинг 12 – Код создания индекса IX\_TransactionTypes для таблицы TransactionTypes*

В результате выполнения кода был создан новый индекс.



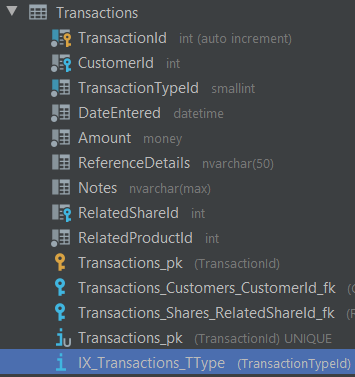
*Рисунок 56 – Создан новый индекс в таблице TransactionTypes*

Был создан новый индекс *IX\_Transactions\_TType* с использованием команд T-SQL.

USE ApressFinancial  
GO  
  
CREATE NONCLUSTERED INDEX IX\_Transactions\_TType  
ON TransactionDetails.Transactions (TransactionTypeId ASC)  
WITH ( STATISTICS\_NORECOMPUTE = OFF, SORT\_IN\_TEMPDB = ON,  
 DROP\_EXISTING = OFF, IGNORE\_DUP\_KEY = OFF, ONLINE = ON,  
 ALLOW\_ROW\_LOCKS = ON, ALLOW\_PAGE\_LOCKS = ON) ON [SECONDARY]  
GO

*Листинг 13 – Код создания индекса IX\_Transaction\_TType для таблицы Transactions*

В результате выполнения кода был создан новый индекс.



*Рисунок 57 – Создан новый индекс в таблице Transactions*

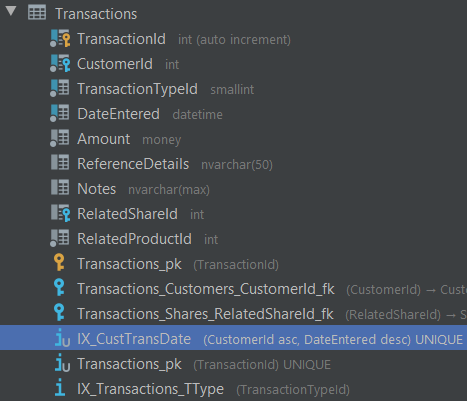
### Изменение индекса.

Изменить индекс с использованием инструкции ALTER невозможно. Чтобы изменить индекс, необходимо удалить старый и создать новый. Можно пересоздать уже существующий индекс. Для этого при создании индекса необходимо указать параметр DROP\_EXISTING со значением ON.

Далее был создан ещё один индекс *IX\_CustTransDate*.

USE ApressFinancial  
GO  
  
CREATE UNIQUE CLUSTERED INDEX IX\_CustTransDate  
ON TransactionDetails.Transactions (CustomerId ASC, DateEntered DESC)  
WITH ( STATISTICS\_NORECOMPUTE = OFF, SORT\_IN\_TEMPDB = ON,  
 DROP\_EXISTING = OFF, IGNORE\_DUP\_KEY = OFF, ONLINE = OFF,  
 ALLOW\_ROW\_LOCKS = ON, ALLOW\_PAGE\_LOCKS = OFF ) ON [SECONDARY]  
GO

*Листинг 14 – Код создания индекса IX\_CustTransDate*



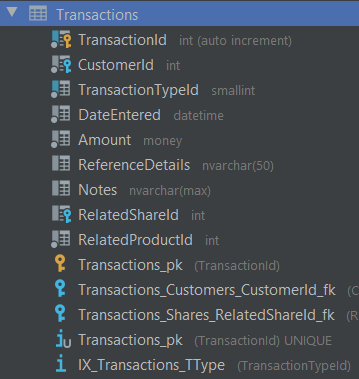
*Рисунок 58 – Создан новый индекс в таблице Transactions*

### Удаление индекса.

Был удалён индекс *IX\_CustTransDate*.

USE ApressFinancial  
GO  
  
DROP INDEX IX\_CustTransDate  
ON TransactionDetails.Transactions  
GO

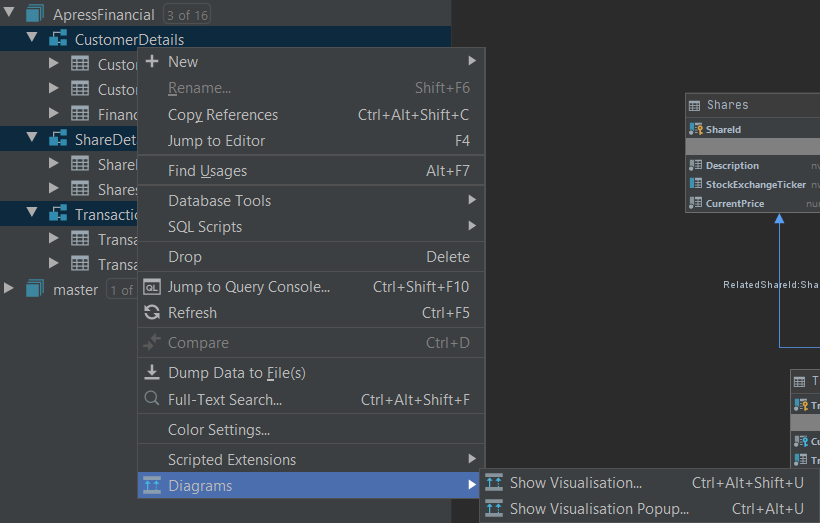
*Листинг 15 – Код удаления индекса IX\_CustTransDate*



*Рисунок 59 – Индекс удалён*

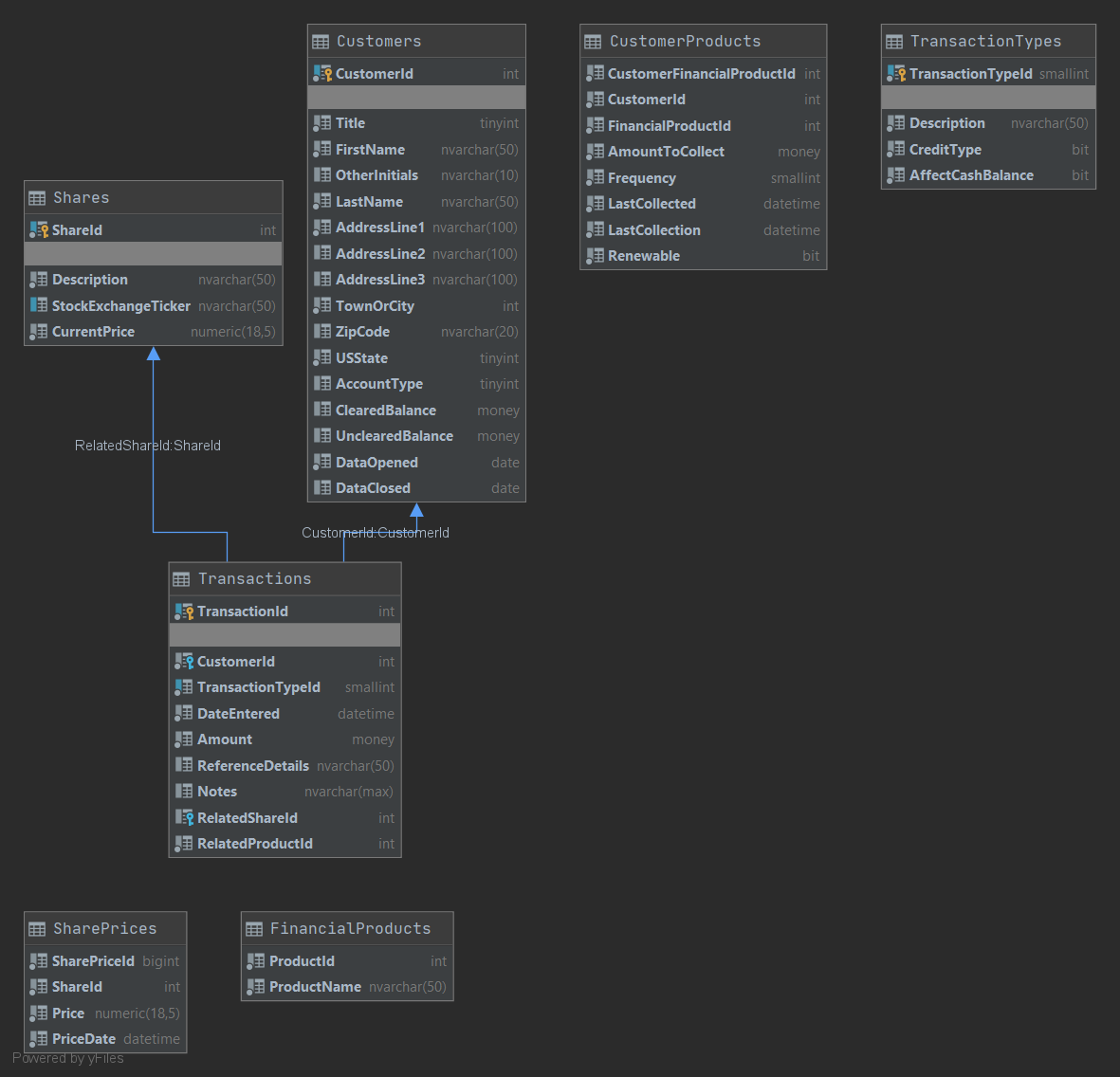
### Построение диаграмм баз данных.

DataGrip IDE уже имеет необходимые инструменты для построения диаграмм. Для этого используется пакет yFiles.



*Рисунок 60 – Создание диаграммы через диалоговое окно*

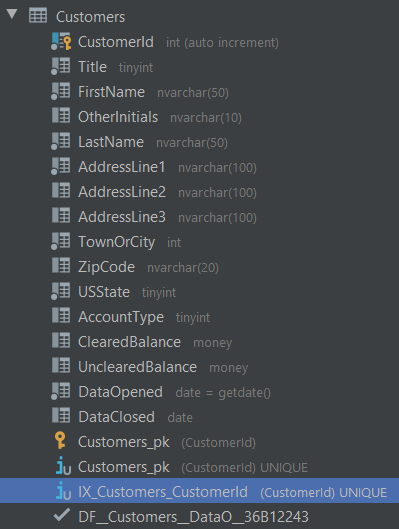
В результате была получена диаграмма для текущей учебной базы данных.



*Рисунок 61 – Диаграмма базы данных*

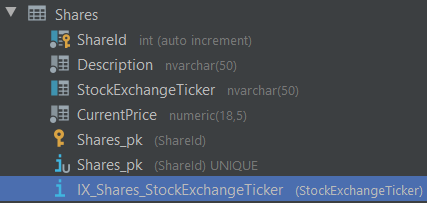
## Окончательные результаты.

### С помощью контекстного меню был создан индекс *IX\_Customers\_CustomerId*.

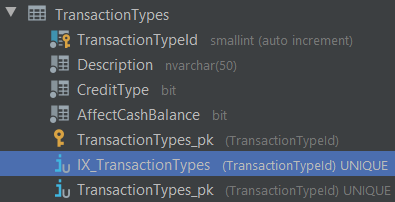


*Рисунок 62 – Новый индекс был добавлен в таблицу Customers*

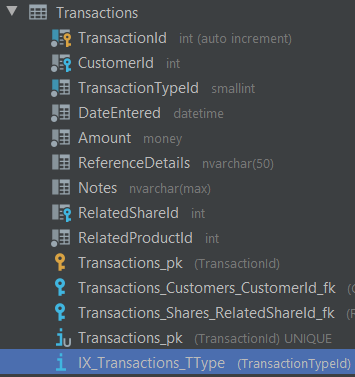
### Были созданы индексы с использованием Query Editor.



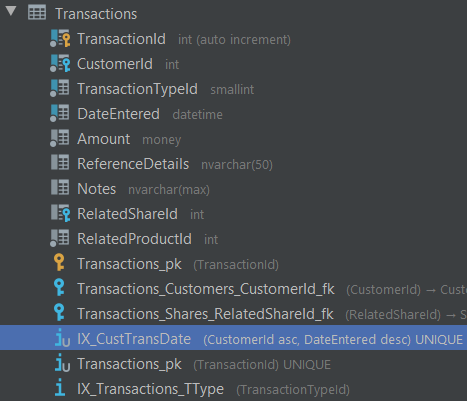
*Рисунок 63 – Создан новый индекс в таблице Shares*



*Рисунок 64 – Создан новый индекс в таблице TransactionTypes*

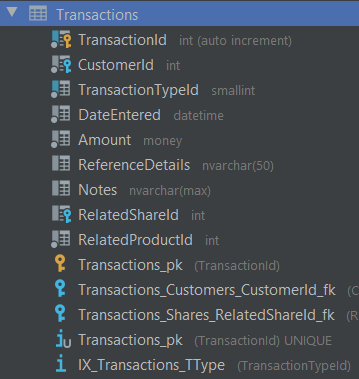


*Рисунок 65 – Создан новый индекс в таблице Transactions*



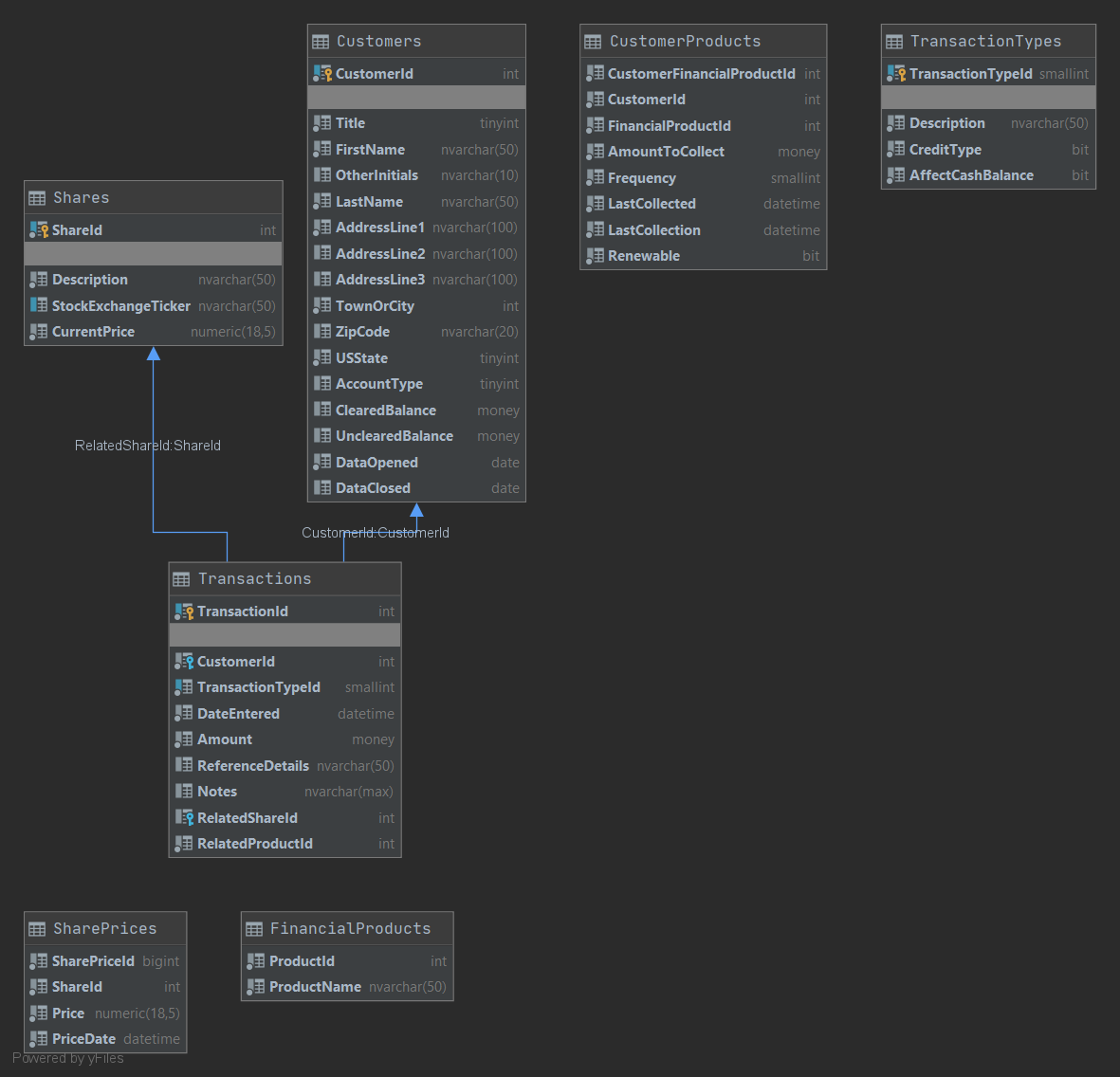
*Рисунок 66 – Создан новый индекс в таблице Transactions*

### Был удалён индекс с использованием Query Editor.



*Рисунок 67 – Индекс удалён*

### Была построена диаграмма для базы данных.



*Рисунок 68 – Диаграмма базы данных*

## Выводы и анализ результатов работы.

В ходе данной лабораторной работы локально были созданы индексы для таблиц базы данных с помощью контекстного меню DataGrip IDE и с помощью SQL запроса. Был рассмотрен процесс построения диаграммы для базы данных.

# «Обработка данных»

## Цель работы.

Рассмотреть различные сценарии работы с данными с помощью SQL запросов.

## Задачи, решаемые при выполнении работы.

### Добавление данных.

### Обновление данных.

### Удаление данных.

## Объект исследования.

Microsoft SQL Server, учебная база данных.

## Исходные данные.

### Методическое пособие.

### Учебная база данных.

## Выполнение работы.

### Добавление данных.

Перед началом работы был установлен строковый формат даты.

USE ApressFinancial  
GO  
  
SET DATEFORMAT dmy  
GO

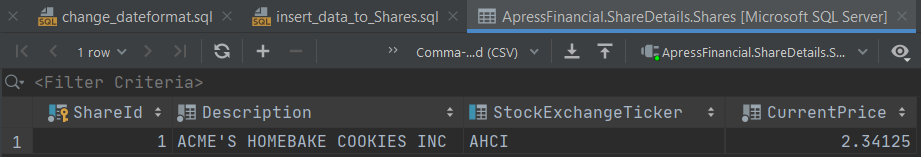
*Листинг 16 – Код установки строкового формата даты*

В таблицу *Shares* были вставлены новые данные.

USE ApressFinancial  
GO  
  
INSERT INTO [ShareDetails].[Shares]  
 ([Description],  
 [StockExchangeTicker],  
 [CurrentPrice])  
 VALUES  
 ('ACME''S HOMEBAKE COOKIES INC',  
 'AHCI',  
 2.34125)  
GO

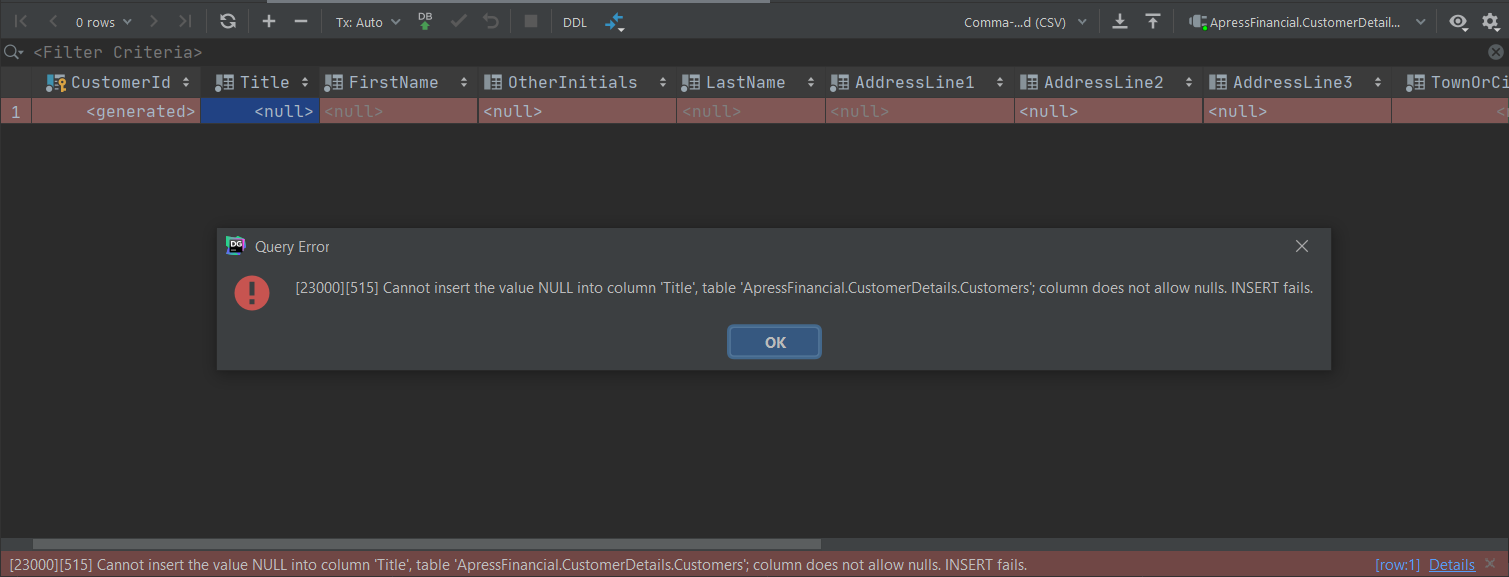
*Листинг 17 – Код для вставки новых данных*

Данные были успешно добавлены в таблицу.



*Рисунок 69 – Данные в таблице Shares*

Была открыта таблица *Customers*. В данную таблицу не получается добавить строку с пустыми полями. Возникает ошибка, так как некоторые поля не поддерживают значение null.



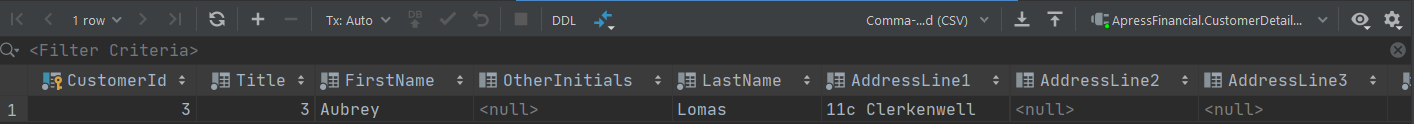
*Рисунок 70 – Добавить новое поле без данных невозможно*

В таблицу Customers были добавлены данные с помощью Query Editor.

USE ApressFinancial  
GO  
  
INSERT INTO CustomerDetails.Customers  
(Title, LastName, FirstName, OtherInitials, AddressLine1, TownOrCity,  
 USState, ZipCode, AccountType, ClearedBalance, UnclearedBalance)  
VALUES (3, 'Lomas', 'Aubrey', NULL, '11c Clerkenwell', 4, 8, 0, 2, 437.97, -10.56)  
GO

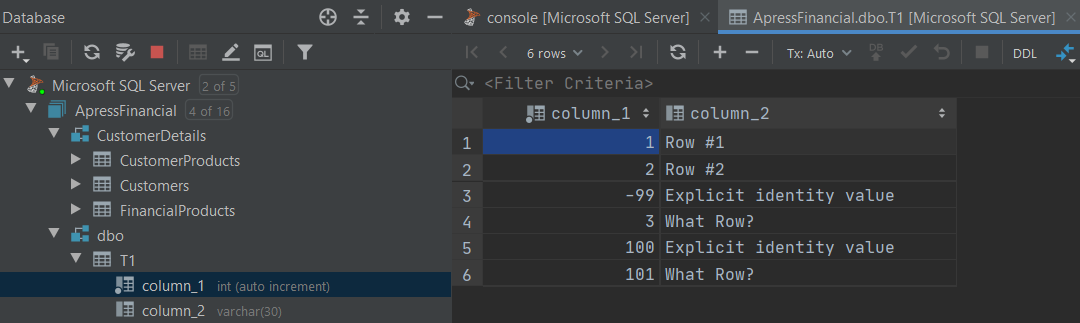
*Листинг 18 – Код вставки данных в таблицу Customers*

В результате в таблицу Customers были вставлены новые данные. При этом значения в полях CustomerId и DataOpened были сгенерированы автоматически.



*Рисунок 71 – В таблице Customers появилась новая запись*

Была рассмотрена вставка значений в поле IDENTITY. Для этого был выполнен код из файла SQLQuery\_Insert\_to\_Identity.sql. В результате в базе dbo появилась таблица T1 с двумя столбцами column\_1 и column\_2.



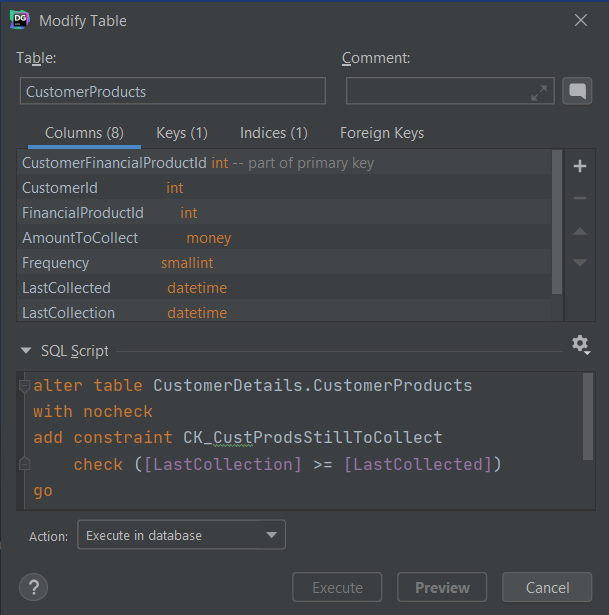
*Рисунок 72 – Результат выполнения файла SQL*

Были добавлены новые ограничения для таблицы CustomerProducts.

USE ApressFinancial  
GO  
  
ALTER TABLE CustomerDetails.CustomerProducts  
ADD CONSTRAINT PK\_CustomerProducts  
PRIMARY KEY CLUSTERED (CustomerFinancialProductId)  
ON [SECONDARY]  
GO  
  
ALTER TABLE CustomerDetails.CustomerProducts  
WITH NOCHECK  
ADD CONSTRAINT CK\_CustProds\_AmtCheck  
CHECK (AmountToCollect > 0)  
GO  
  
ALTER TABLE CustomerDetails.CustomerProducts  
WITH NOCHECK  
ADD CONSTRAINT DF\_CustProd\_Renewable  
DEFAULT (0) FOR Renewable  
GO

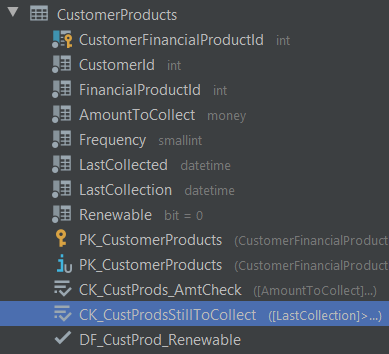
*Листинг 19 – Создание новых ограничений для столбцов в таблице CustomerProducts*

Ещё одно ограничение было добавлено с помощью окна Modify Table…



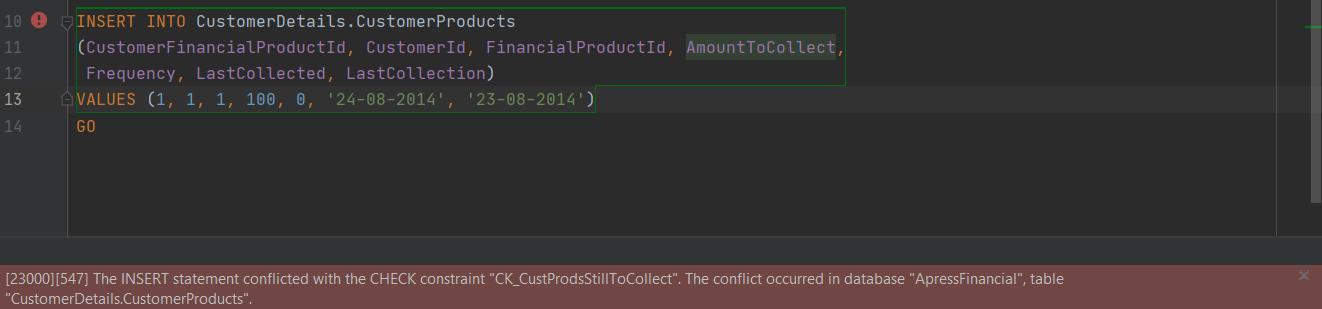
*Рисунок 73 – Окно Modify Table*

В открывшемся окне были написаны команды для создания ограничения для столбцов LastCollected и LastCollection.



*Рисунок 74 – Новое ограничение было создано*

Ограничения были проверены на практике. Не удалось вставить запись, где LastCollection был меньше, чем LastCollected.

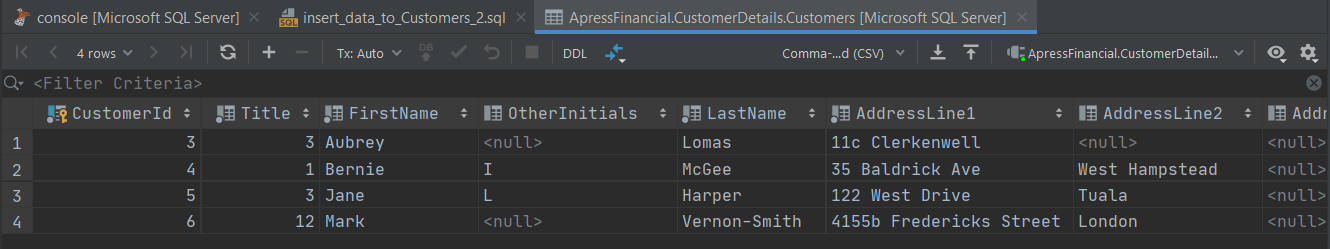


*Рисунок 75 – Ошибка вставки данных*

В таблицу Customers было вставлено несколько записей с помощью одной команды INSERT.

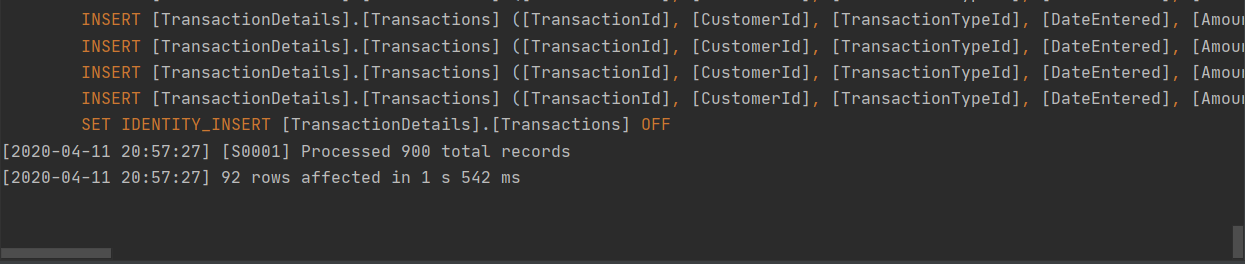
USE ApressFinancial  
GO  
  
INSERT INTO CustomerDetails.Customers  
(Title, FirstName, OtherInitials, LastName, AddressLine1, AddressLine2,  
 TownOrCity, USState, ZipCode, ClearedBalance, UnclearedBalance)  
VALUES  
(1, 'Bernie', 'I', 'McGee', '35 Baldrick Ave', 'West Hampstead',  
 18651, 0, 0, 6653.11, 0.00),  
(3, 'Jane', 'L', 'Harper', '122 West Drive', 'Tuala',  
 185111, 14, 111, 53.11, 27.50),  
(12, 'Mark', NULL, 'Vernon-Smith', '4155b Fredericks Street', 'London',  
 157, 0, 97, -751, 900.00)  
GO

*Листинг 20 – Код вставки нескольких записей*



*Рисунок 76 – Новые записи появились в таблице Customers*

Учебная база данных была заполнена записями с помощью скрипта SQLQuery\_BulkInsert.sql.



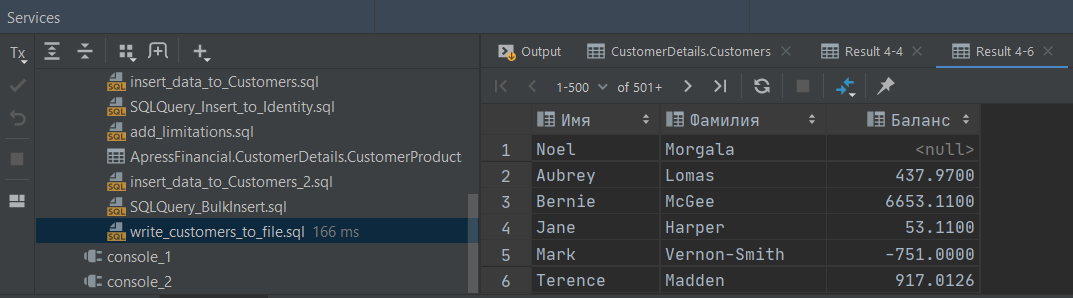
*Рисунок 77 – Скрипт успешно выполнен*

Данные из таблицы Customers были выведены на экран.

USE ApressFinancial  
GO  
  
SELECT \* FROM CustomerDetails.Customers  
GO  
  
USE ApressFinancial  
GO  
  
SELECT FirstName, LastName, ClearedBalance  
FROM CustomerDetails.Customers  
GO  
  
USE ApressFinancial  
GO  
  
SELECT FirstName AS N'Имя',  
 LastName AS N'Фамилия',  
 ClearedBalance AS N'Баланс'  
FROM CustomerDetails.Customers  
GO

*Листинг 21 – Код для вывода записей на экран*

DataGrip IDE выводит записи в удобном формате csv таблиц. Если необходимо, их можно экспортировать из программы на диск.



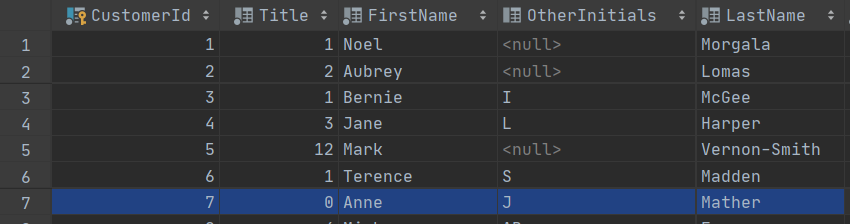
*Рисунок 78 – Результат выполнения кода*

### Обновление данных.

Было изменено значение поля LastName с помощью следующего кода.

USE ApressFinancial  
GO  
  
UPDATE CustomerDetails.Customers  
SET LastName = 'Mather'  
WHERE CustomerId = 7  
GO

*Листинг 22 – Код обновления данных*

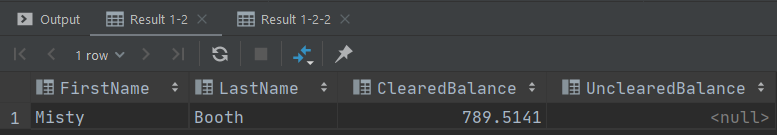


*Рисунок 79 – Данные обновились*

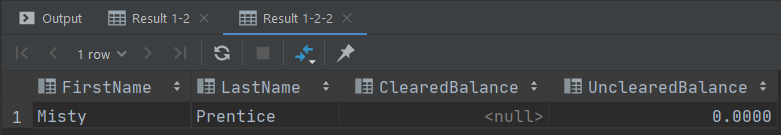
Были обновлены данные столбцов с использованием значений из других столбцов.

USE ApressFinancial  
GO  
  
SELECT FirstName, LastName, ClearedBalance, UnclearedBalance  
FROM CustomerDetails.Customers  
WHERE LastName = 'Booth';  
  
DECLARE @ValueToUpdate NVARCHAR(50)  
SET @ValueToUpdate = 'Prentice';  
  
UPDATE CustomerDetails.Customers  
SET LastName = @ValueToUpdate,  
 ClearedBalance = ClearedBalance + UnclearedBalance,  
 UnclearedBalance = 0  
WHERE LastName = 'Booth';  
  
SELECT FirstName, LastName, ClearedBalance, UnclearedBalance  
FROM CustomerDetails.Customers  
WHERE LastName = 'Prentice'  
GO

*Листинг 23 – Код обновления данных*



*Рисунок 80 – Данные до обновления*

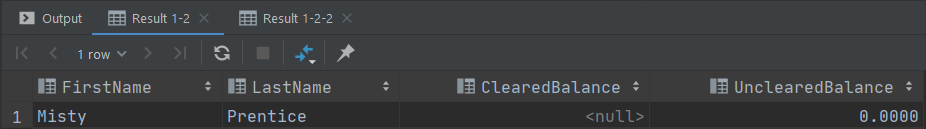


*Рисунок 81 – Данные после обновления*

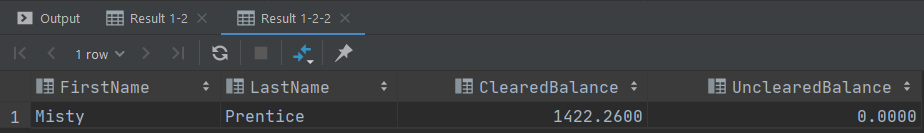
Проверим неявное преобразование данных. Для этого выполним следующий код.

USE ApressFinancial  
GO  
  
SELECT FirstName, LastName, ClearedBalance, UnclearedBalance  
FROM CustomerDetails.Customers  
WHERE LastName = 'Prentice';  
  
DECLARE @WrongDataType NVARCHAR(50)  
SET @WrongDataType = '1422.26';  
  
UPDATE CustomerDetails.Customers  
SET ClearedBalance = @WrongDataType  
WHERE LastName = 'Prentice';  
  
SELECT FirstName, LastName, ClearedBalance, UnclearedBalance  
FROM CustomerDetails.Customers  
WHERE LastName = 'Prentice';  
GO

*Листинг 24 – Код обновления данных, в котором присутствует неявное преобразование типов*



*Рисунок 82 – Данные до обновления*

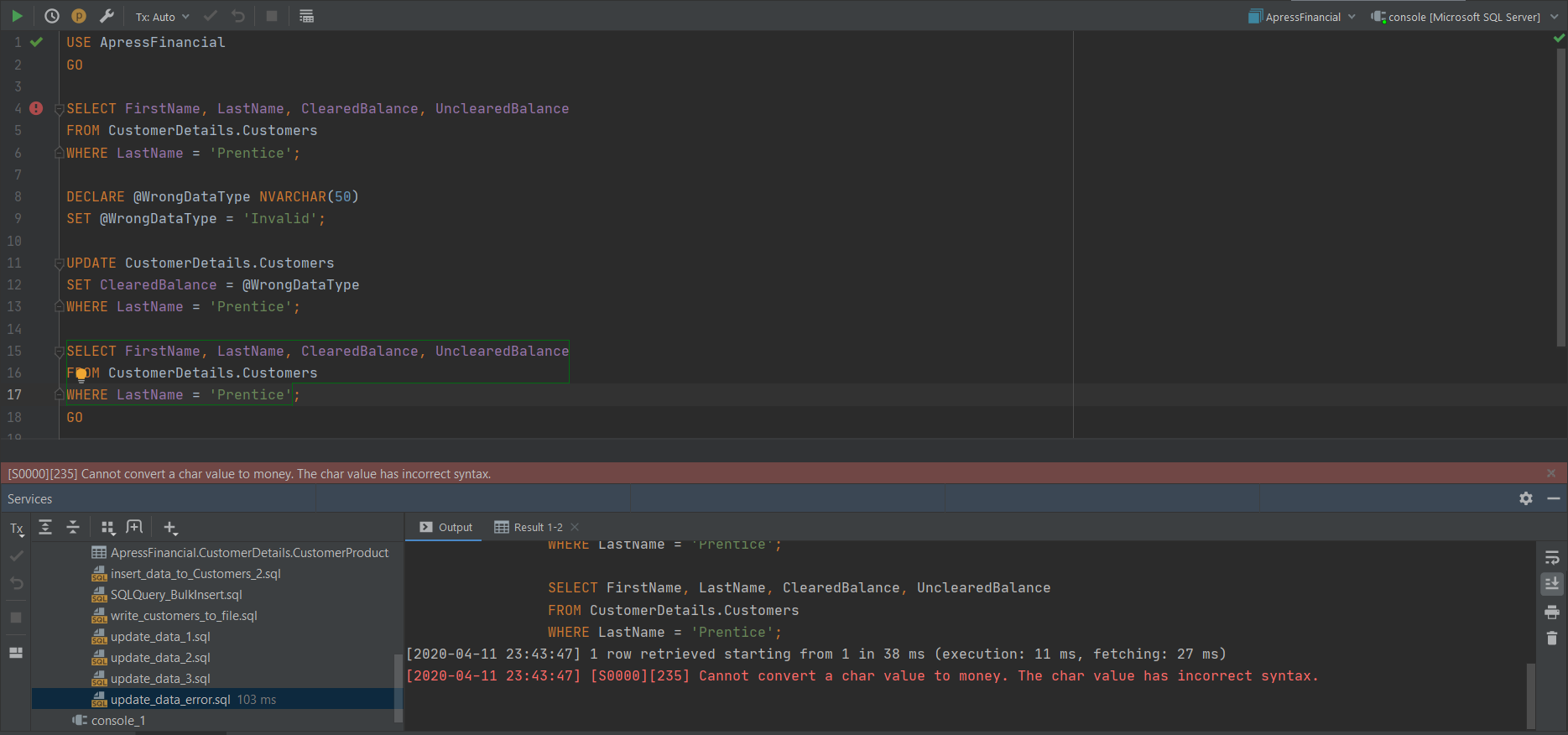


*Рисунок 83 – Данные после обновления*

Невозможно неявно преобразовать текст к числовому значению.

USE ApressFinancial  
GO  
  
SELECT FirstName, LastName, ClearedBalance, UnclearedBalance  
FROM CustomerDetails.Customers  
WHERE LastName = 'Prentice';  
  
DECLARE @WrongDataType NVARCHAR(50)  
SET @WrongDataType = 'Invalid';  
  
UPDATE CustomerDetails.Customers  
SET ClearedBalance = @WrongDataType  
WHERE LastName = 'Prentice';  
  
SELECT FirstName, LastName, ClearedBalance, UnclearedBalance  
FROM CustomerDetails.Customers  
WHERE LastName = 'Prentice';  
GO

*Листинг 25 – Код обновления данных с ошибкой*



*Рисунок 84 – Выполнение кода вызывает ошибку*

### Удаление данных.

Во временную базу данных были импортированы данные из таблицы Customers.

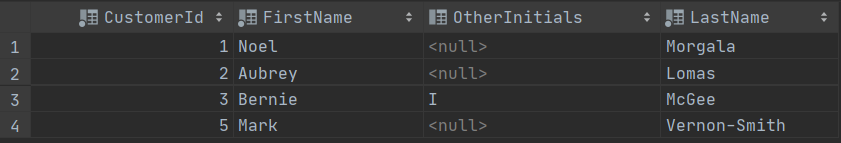
USE tempdb  
GO  
  
SELECT TOP (5) CustomerId, FirstName, OtherInitials, LastName  
INTO dbo.CustTemp  
FROM ApressFinancial.CustomerDetails.Customers  
GO

*Листинг 26 – Импорт данных из таблицы Customers*

Была удалена запись с CustomerId = 4.

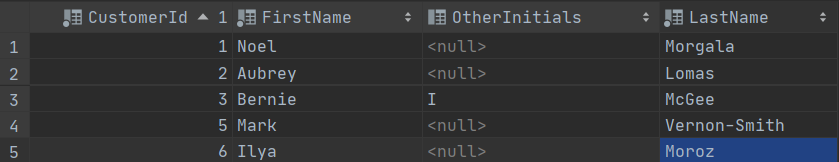
USE tempdb  
GO  
  
DELETE FROM dbo.CustTemp  
WHERE CustomerId = 4  
GO

*Листинг 27 – Удаление записи с CustomerId = 4*



*Рисунок 85 – Запись успешно удалена*

В таблицу CustTemp была добавлена новая запись.

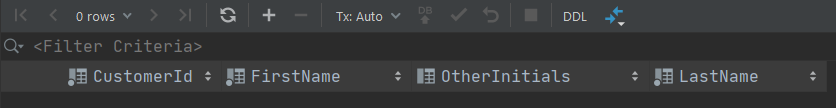


*Рисунок 86 – Новая запись была добавлена*

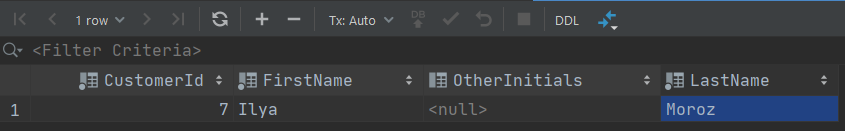
Были удалены все данные из таблицы CustTemp.

USE tempdb  
GO  
  
DELETE FROM dbo.CustTemp  
GO

*Листинг 28 – Код удаления всех данных из таблицы CustTemp*



*Рисунок 87 – Таблица CustTemp пустая*

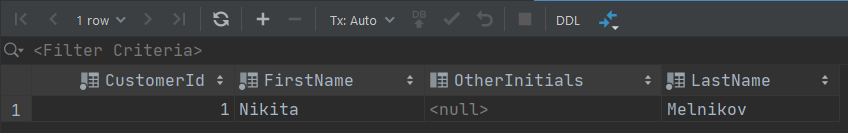


*Рисунок 88 – Создание новой записи. Значение CustomerId автоматически 7.*

Сделаем так, чтобы поле CustomerId начиналось с 1.

USE tempdb  
GO  
  
TRUNCATE TABLE dbo.CustTemp  
GO

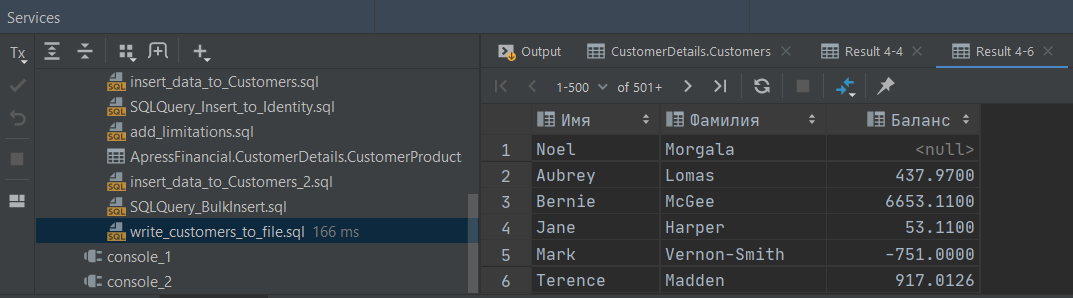
*Листинг 29 – Задача полям IDENTITY начального значения*



*Рисунок 89 – Теперь новая запись имеет значение 1 в поле CustomerId*

## Окончательные результаты.

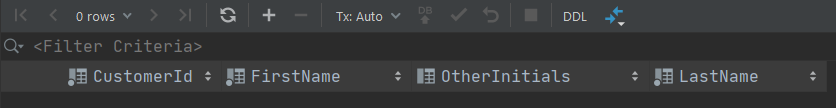
### Новые данные были добавлены в базу данных.



*Рисунок 90 – Результат выполнения кода*

### Были отработаны различные сценарии обновления данных в базе данных.

### Во временную базу данных были импортированы данные, которые впоследствии были удалены.



*Рисунок 91 – Таблица CustTemp пустая*

## Выводы и анализ результатов работы.

В ходе данной лабораторной работы были отработаны различные варианты взаимодействия с данными в базе данных. Были добавлены новые записи, записи были обновлены, в конце часть данных была импортирована во временную базу данных и удалена.

# «Построение представлений»

## Цель работы.

Построить различные представления для работы с учебной базой данных.

## Задачи, решаемые при выполнении работы.

### Построение представлений.

## Объект исследования.

Microsoft SQL Server, учебная база данных.

## Исходные данные.

### Методическое пособие.

### Учебная база данных.

## Выполнение работы.

### Построение представления ShareDetails.v\_CurrentShares.

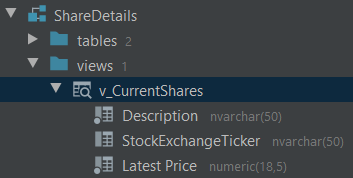
К сожалению, DataGrip не располагает функционалом для создания представлений с помощью контекстного меню. Поэтому, в данной лабораторной работе создание представлений было рассмотрено только с помощью SQL запросов.

Для создания представления ShareDetails.v\_CurrentShares был создан файл *create\_v\_CurrentShares.sql* со следующим исходным кодом:

USE ApressFinancial  
GO  
  
CREATE VIEW ShareDetails.v\_CurrentShares  
AS  
SELECT TOP(100) Description, StockExchangeTicker, CurrentPrice AS [Latest Price]  
FROM ShareDetails.Shares  
WHERE (CurrentPrice > 0)  
ORDER BY Description

*Листинг 30 – Создание представления ShareDetails.v\_CurrentShares*

Код был успешно выполнен, в структуре БД появилось новое представление.



*Рисунок 92 – Новое представление*

### Создание представления с помощью представления

Необходимо добавить столбец ShareId в существующее представление ShareDetails.v\_CurrentShares. Для этого был выполнен следующий код:

USE ApressFinancial  
GO  
  
ALTER VIEW ShareDetails.v\_CurrentShares  
AS  
SELECT TOP(100) ShareId, Description, StockExchangeTicker, CurrentPrice AS [Latest Price]  
FROM ShareDetails.Shares  
WHERE (CurrentPrice > 0)  
ORDER BY Description

*Листинг 31 – Содержимое файла update\_v\_CurrentShares.sql*

Необходимо создать новое представление ShareDetails.v\_SharePrices.

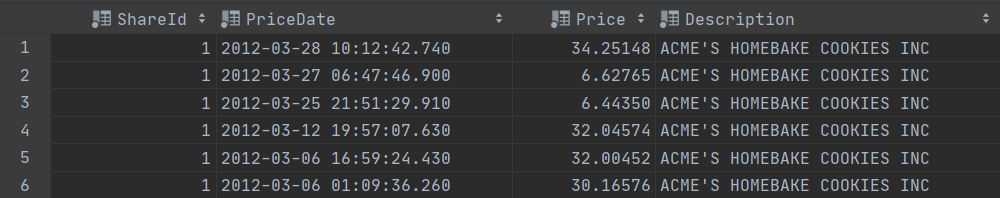
USE ApressFinancial  
GO  
  
CREATE VIEW ShareDetails.v\_SharePrices  
AS  
SELECT TOP(100) PERCENT ShareDetails.SharePrices.ShareId, ShareDetails.SharePrices.PriceDate, ShareDetails.SharePrices.Price, ShareDetails.v\_CurrentShares.Description  
FROM ShareDetails.SharePrices INNER JOIN  
 ShareDetails.v\_CurrentShares ON ShareDetails.SharePrices.ShareId = ShareDetails.v\_CurrentShares.ShareId  
ORDER BY ShareDetails.v\_CurrentShares.Description, ShareDetails.SharePrices.Price DESC

*Листинг 32 – Код создания нового представления*

Новое представление связано с существующим представлением ShareDetails.v\_CurrentShares. Протестируем только что созданное представление с помощью следующего кода:

USE ApressFinancial  
GO  
  
SELECT \* FROM ShareDetails.v\_SharePrices  
GO

*Листинг 33 – Вывод всех строк из представления ShareDetails.v\_SharePrices*

**

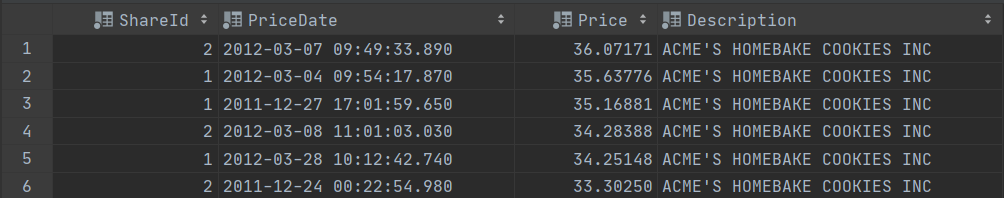
*Рисунок 93 – Вывод всех полей в консоль*

Представление ShareDetails.v\_SharePrices было изменено так, чтобы записи сортировались по двум столбикам: Price и Description - с помощью следующего кода:

USE ApressFinancial  
GO  
  
ALTER VIEW ShareDetails.v\_SharePrices  
AS  
SELECT  
 p.ShareId,  
 p.PriceDate,  
 p.Price,  
 c.[Description]  
FROM ShareDetails.SharePrices AS p  
INNER JOIN ShareDetails.v\_CurrentShares AS c  
 ON p.ShareId = c.ShareId  
ORDER BY  
 c.[Description],  
 p.Price DESC  
OFFSET (0) ROWS  
GO

*Листинг 34 – Изменение нового представления*

Теперь сортировка работает правильно:



*Рисунок 94 – Сортировка работает правильно*

### Создание оставшихся представлений

С помощью SQL-запросов было создано новое представление CustomerDetails.v\_CustTrans.

USE ApressFinancial  
GO  
  
CREATE VIEW CustomerDetails.v\_CustTrans  
AS  
SELECT c.CustomerId, c.FirstName, c.OtherInitials, tt.[Description],  
 t.DateEntered, t.Amount, t.ReferenceDetails  
FROM CustomerDetails.Customers AS c  
JOIN TransactionDetails.Transactions AS t  
 ON t.CustomerId = c.CustomerId  
JOIN TransactionDetails.TransactionTypes AS tt  
 ON tt.TransactionTypeId = t.TransactionTypeId  
WHERE DateEntered >= '01-01-2012'  
WITH CHECK OPTION  
GO

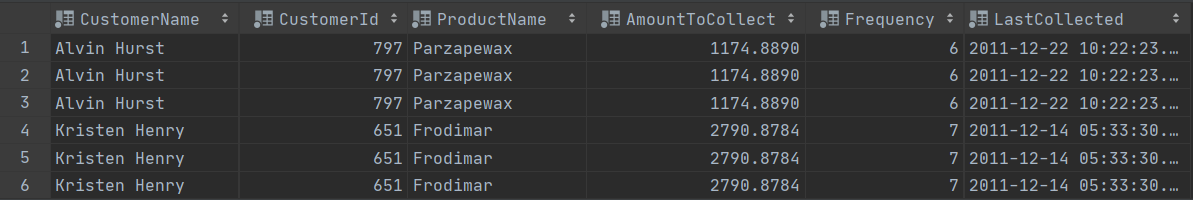
*Листинг 35 – Код создания представления CustomerDetails.v\_CustTrans*

Было создано новое представление CustomerDetails.v\_CustFinProducts с использованием параметра SCHEMABINDING.

USE ApressFinancial  
GO  
  
IF *EXISTS*( SELECT TABLE\_NAME FROM INFORMATION\_SCHEMA.VIEWS  
 WHERE TABLE\_NAME = N'v\_CustFinProducts'  
 AND TABLE\_SCHEMA = N'CustomerDetails' )  
DROP VIEW CustomerDetails.v\_CustFinProducts  
GO  
  
CREATE VIEW CustomerDetails.v\_CustFinProducts  
WITH SCHEMABINDING  
AS  
SELECT c.FirstName + ' ' + c.LastName AS CustomerName, c.CustomerId,  
 fp.ProductName, cp.AmountToCollect, cp.Frequency, cp.LastCollected  
FROM CustomerDetails.Customers AS c  
JOIN CustomerDetails.CustomerProducts AS cp  
 ON cp.CustomerId = c.CustomerId  
JOIN CustomerDetails.FinancialProducts AS fp  
 ON fp.ProductId = cp.FinancialProductId  
GO

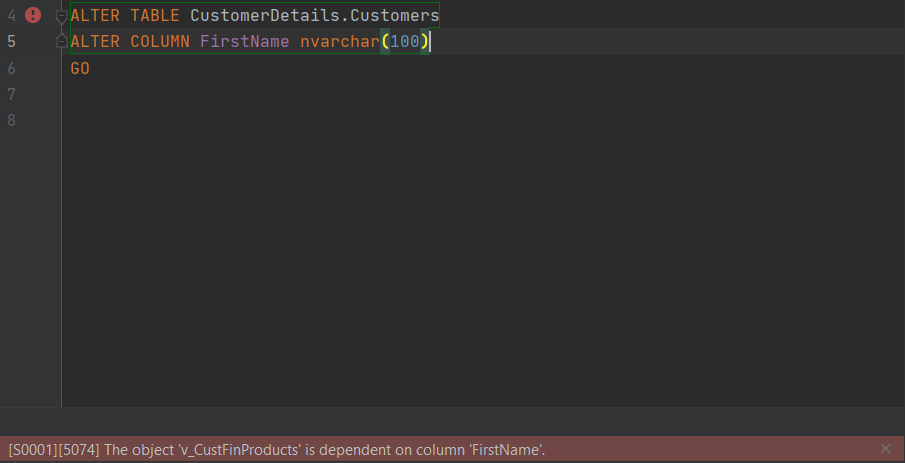
*Листинг 36 – Код создания представления CustomerDetails.v\_CustFinProducts*

С помощью отработанного в предыдущем задании запроса представление было протестировано.



*Рисунок 95 – Содержимое представления CustomerDetails.v\_CustFinProducts*

Попытка изменить поле в CustomerDetails.Customers привела к ошибке:

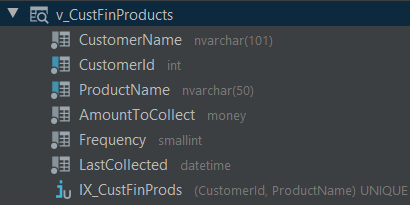
****

*Рисунок 96 – Ошибка при попытке изменить поле*

Представление CustomerDetails.v\_CustFinProducts было проиндексировано с помощью следующего кода:

USE ApressFinancial  
GO  
  
SET ANSI\_NULLS ON  
SET ANSI\_PADDING ON  
SET ANSI\_WARNINGS ON  
SET CONCAT\_NULL\_YIELDS\_NULL ON  
SET ARITHABORT ON  
SET QUOTED\_IDENTIFIER ON  
SET NUMERIC\_ROUNDABORT OFF  
GO  
  
CREATE UNIQUE CLUSTERED INDEX IX\_CustFinProds  
ON CustomerDetails.v\_CustFinProducts (CustomerId, ProductName)  
GO  
  
SET ANSI\_NULLS OFF  
SET ANSI\_PADDING OFF  
SET ANSI\_WARNINGS OFF  
SET CONCAT\_NULL\_YIELDS\_NULL OFF  
SET ARITHABORT OFF  
SET QUOTED\_IDENTIFIER OFF  
SET NUMERIC\_ROUNDABORT ON  
GO

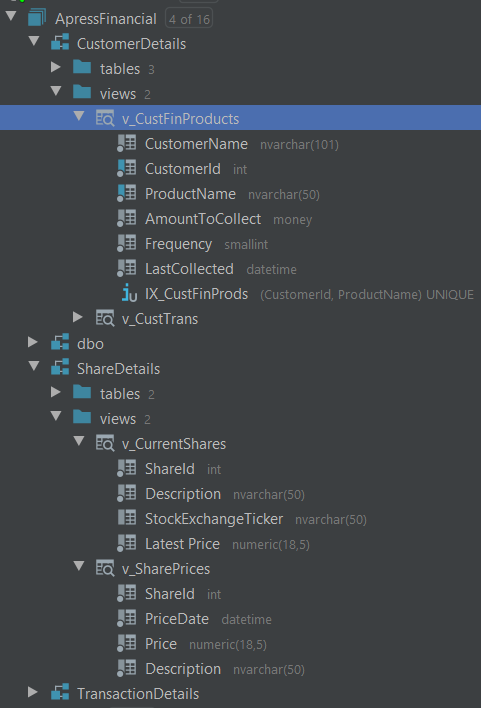
*Листинг 37 – Создание индекса IX\_CustFinProds*



*Рисунок 97 – В представлении появился новый индекс*

## Окончательные результаты.

### В учебной базе данных были созданы новые представления. Новая структура базы данных выглядит следующим образом:



*Рисунок 98 – Новая структура учебной БД*

## Выводы и анализ результатов работы.

В ходе данной лабораторной работы были рассмотрены представления в языке SQL. Они необходимы для создания «виртуальных таблиц», которые имеют ряд преимуществ над обычными: динамически извлекают данные из базы данных, позволяют возвращать отформатированные значения из таблиц в нужной и удобной форме, защищают данные, так как представления могут дать доступ к части таблицы, а не ко всей таблице.

# «Реализация пользовательских функций и хранимых процедур»

## Цель работы.

Изучить создание и возможности пользовательских функций и хранимых процедур на языке SQL.

## Задачи, решаемые при выполнении работы.

### Создание пользовательских функций.

### Создание хранимых процедур.

## Объект исследования.

Microsoft SQL Server, учебная база данных.

## Исходные данные.

### Методическое пособие.

### Учебная база данных.

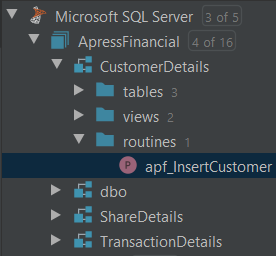
## Выполнение работы.

### Создание хранимых процедур в DataGrip возможно только с использованием SQL редактора. Был написан следующий SQL-запрос:

CREATE PROCEDURE CustomerDetails.*apf\_InsertCustomer* @CustTitle int,  
 @FirstName nvarchar(50),  
 @CustInitials nvarchar(10),  
 @LastName nvarchar(50),  
 @AddressLine1 nvarchar(100),  
 @AddressLine2 nvarchar(100),  
 @AddressLine3 nvarchar(300),  
 @TownOrCity int,  
 @ZipCode nvarchar(20),  
 @USState tinyint,  
 @AccountTypeId tinyint,  
 @ClearedBalance money  
AS  
BEGIN  
 SET NOCOUNT ON;  
 INSERT INTO CustomerDetails.Customers  
 (Title, FirstName, OtherInitials, LastName,  
 AddressLine1, AddressLine2, AddressLine3, TownOrCity,  
 ZipCode, USState, AccountType, ClearedBalance)  
 VALUES  
 (@CustTitle, @FirstName, @CustInitials, @LastName,  
 @AddressLine1, @AddressLine2, @AddressLine3, @TownOrCity,  
 @ZipCode, @USState, @AccountTypeId, @ClearedBalance)  
END  
GO

*Листинг 38 – Код создания процедуры InsertCustomer*

Была создана новая хранимая процедура для добавления новых Customers.



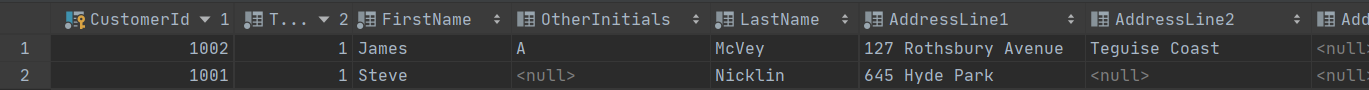
*Рисунок 99 – Хранимая процедура добавлена в БД*

Работа процедуры была проверена с помощью следующего кода в консоли SQL:

CustomerDetails.*apf\_InsertCustomer* 1, 'Steve', NULL,  
 'Nicklin', '645 Hyde Park', NULL, NULL,  
 7545, 'W1A 1AA', 0, 1, 0  
GO  
EXEC CustomerDetails.*apf\_InsertCustomer* @CustTitle = 1, @FirstName = 'James',  
 @CustInitials = 'A', @LastName = 'McVey', @AddressLine1 = '127 Rothsbury Avenue',  
 @AddressLine2 = 'Teguise Coast', @AddressLine3 = NULL, @TownOrCity = 94511,  
 @ZipCode = '121100', @USState = 0, @AccountTypeId = 2, @ClearedBalance = 0  
GO  
  
SELECT \* FROM CustomerDetails.Customers ORDER BY CustomerId OFFSET 1000 ROWS  
GO

*Листинг 39 – Использование процедуры для создания новых Customers*

В консоль были выведены данные из таблицы Customers, там появились новые записи:



*Рисунок 100 – Содержимое таблицы Customers*

Была создана процедура для расчёта сальдо по входным параметрам.

CREATE PROCEDURE CustomerDetails.*apf\_CustMovement* @CustId int,  
 @FromDate datetime,  
 @ToDate datetime  
AS  
BEGIN  
 DECLARE @RunningBal money, @StillCalc bit, @LastTran int  
 SELECT @StillCalc = 1, @LastTran = 0, @RunningBal = 0  
 WHILE @StillCalc = 1  
 BEGIN  
 SELECT TOP 1  
 @RunningBal = @RunningBal +  
 CASE  
 WHEN tt.CreditType = 1 THEN t.Amount  
 ELSE t.Amount \* -1  
 END,  
 @LastTran = t.TransactionId  
 FROM CustomerDetails.Customers AS c  
 JOIN TransactionDetails.Transactions AS t  
 ON t.CustomerId = c.CustomerId  
 JOIN TransactionDetails.TransactionTypes AS tt  
 ON tt.TransactionTypeId = t.TransactionTypeId  
 WHERE t.TransactionId > @LastTran  
 AND tt.AffectCashBalance = 1  
 AND DateEntered BETWEEN @FromDate AND @ToDate  
 ORDER BY DateEntered  
 IF *@@ROWCOUNT* > 0  
 CONTINUE  
 ELSE  
 BREAK  
 END  
 SELECT @RunningBal AS 'EndBalance'  
END  
GO

*Листинг 40 – Создание процедуры расчёта сальдо*

Созданная процедура была проверена с помощью следующего кода:

INSERT INTO TransactionDetails.Transactions (CustomerId, TransactionTypeId, DateEntered, Amount, RelatedProductId)  
VALUES (6 ,1, '01-08-2015', 100.00, 1),  
 (6, 1, '03-08-2015', 75.67, 1),  
 (6, 2, '05-08-2015', 35.20, 1),  
 (6, 2, '06-08-2015', 20.00, 1)  
GO  
  
EXEC CustomerDetails.*apf\_CustMovement* 1, '01-08-2015', '31-08-2015'  
GO

*Листинг 41 – Проверка работы процедуры*

В результате работы было получено следующее число:



*Рисунок 101 – Результат работы процедуры*

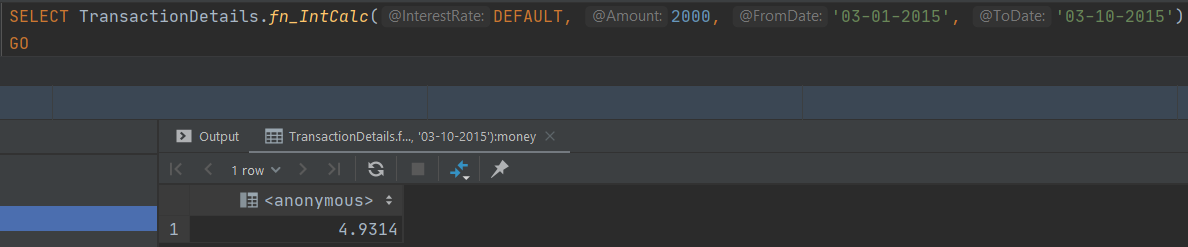
### Создание скалярной пользовательской функции.

Была создана новая скалярная функция, позволяющая рассчитать процент.

CREATE FUNCTION TransactionDetails.*fn\_IntCalc*( @InterestRate numeric(6, 3) = 10, @Amount money,  
 @FromDate date, @ToDate date )  
RETURNS money  
WITH EXECUTE AS CALLER  
AS  
BEGIN  
 DECLARE @IntCalculated money  
 SELECT @IntCalculated = @Amount \* ((@InterestRate / 100.00) \*  
 (*DATEDIFF*(d, @FromDate, @ToDate) / 365.00))  
 RETURN (*ISNULL*(@IntCalculated, 0))  
END  
GO

*Листинг 42 – Код создания функции IntCalc*

Работа функции была проверена следующим образом:



*Рисунок 102 – Проверка скалярной функции*

### Функция, возвращающая табличное значение.

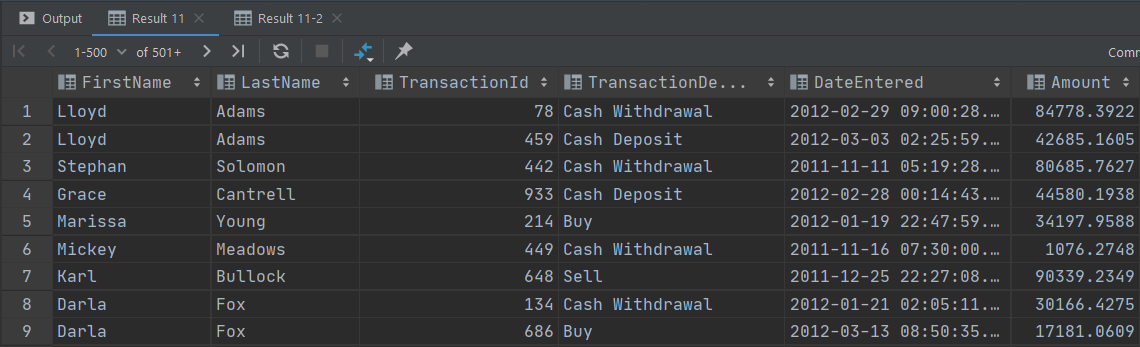
Была создана функция, которая возвращает таблицу строк Transactions.

CREATE FUNCTION TransactionDetails.*ReturnTransaction*(@CustId int)  
RETURNS @Trans TABLE  
( TransactionId int,  
 CustomerId int,  
 TransactionDescription nvarchar(50),  
 DateEntered datetime,  
 Amount money )  
AS  
BEGIN  
 INSERT INTO @Trans  
 SELECT t.TransactionId, t.CustomerId,  
 tt.[Description], t.DateEntered, t.Amount  
 FROM TransactionDetails.Transactions AS t  
 JOIN TransactionDetails.TransactionTypes AS tt  
 ON tt.TransactionTypeId = t.TransactionTypeId  
 WHERE CustomerId = @CustId  
 RETURN  
END  
GO

*Листинг 43 – Код создания функции ReturnTransaction*

SELECT c.FirstName, c.LastName, t.TransactionId,  
 t.TransactionDescription, t.DateEntered, t.Amount  
FROM CustomerDetails.Customers AS c  
CROSS APPLY TransactionDetails.*ReturnTransactions*(c.CustomerId) AS t  
GO  
  
SELECT c.FirstName, c.LastName, t.TransactionId,  
 t.TransactionDescription, t.DateEntered, t.Amount  
FROM CustomerDetails.Customers AS c  
OUTER APPLY TransactionDetails.*ReturnTransactions*(c.CustomerId) AS t  
GO

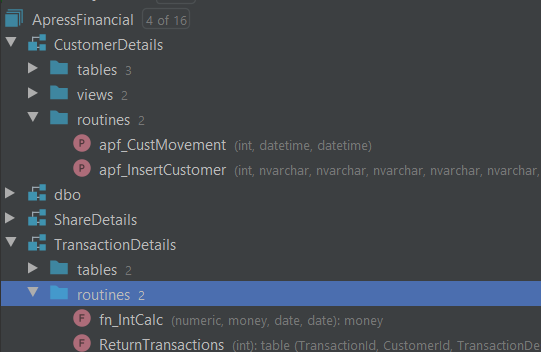
*Листинг 44 – Код проверки функции ReturnTransactions*

**

*Рисунок 103 – Результат работы функции*

## Окончательные результаты.

### Были созданы новые скалярные функции и хранимые процедуры, которые упрощают взаимодействие с базой данных.



*Рисунок 104 – Новая структура базы данных*

## Выводы и анализ результатов работы.

В ходе данной лабораторной работы были рассмотрены скалярные функции и хранимые процедуры. Они позволяют упростить работу с базой данных, ускорить её. В данной лабораторной работе они позволили считать проценты, выводить данные из таблицы, создавать новые данные на основе существующих. Как и в других языках программирования, процедуры и функции являются важнейшей частью синтаксиса T-SQL.

# «Триггеры»

## Цель работы.

Изучить триггеры DML и DDL.

## Задачи, решаемые при выполнении работы.

### Создание триггера DML.

### Создание триггера DDL.

## Объект исследования.

Microsoft SQL Server, учебная база данных.

## Исходные данные.

### Методическое пособие.

### Учебная база данных.

## Выполнение работы.

### Создание триггера DML.

Был создан триггер для вставки данных в таблицу Transactions.

CREATE TRIGGER TransactionDetails.trgInsTransactions  
 ON TransactionDetails.Transactions  
 AFTER INSERT  
AS  
BEGIN  
 UPDATE CustomerDetails.Customers  
 SET ClearedBalance = ClearedBalance +  
 ( SELECT  
 CASE  
 WHEN CreditType = 0 THEN i.Amount \* -1  
 ELSE i.Amount  
 END  
 FROM INSERTED AS i  
 JOIN TransactionDetails.TransactionTypes AS tt  
 ON tt.TransactionTypeId = i.TransactionTypeId  
 WHERE AffectCashBalance = 1 )  
 FROM CustomerDetails.Customers AS c  
 JOIN INSERTED AS i  
 ON i.CustomerId = c.CustomerId  
END  
GO

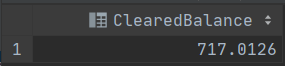
*Листинг 45 – Код создания триггера для вставки данных в таблицу Transactions*

Проверим работу триггера следующим кодом:

SELECT ClearedBalance FROM CustomerDetails.Customers WHERE CustomerId = 6;  
INSERT INTO TransactionDetails.Transactions (CustomerId, TransactionTypeId, Amount, RelatedProductId, DateEntered)  
VALUES (6, 4, 200, 1, *GETDATE*());  
SELECT ClearedBalance FROM CustomerDetails.Customers WHERE CustomerId = 6;  
GO

*Листинг 46 – Код для проверки работы триггера*

Баланс действительно уменьшился на 200.

*Рисунок 105 – Баланс уменьшился на 200*

При ненадёжной транзакции (которую триггер не может обработать) произойдёт ошибка. Так как триггер вернёт значение NULL.

Поэтому изменим триггер следующим кодом:

ALTER TRIGGER TransactionDetails.trgInsTransactions  
 ON TransactionDetails.Transactions  
 AFTER INSERT  
AS  
BEGIN  
 UPDATE CustomerDetails.Customers  
 SET ClearedBalance = ClearedBalance +  
 *ISNULL*((SELECT  
 CASE  
 WHEN CreditType = 0 THEN i.Amount \* -1  
 ELSE i.Amount  
 END  
 FROM INSERTED AS i  
 JOIN TransactionDetails.TransactionTypes AS tt  
 ON tt.TransactionTypeId = i.TransactionTypeId  
 WHERE AffectCashBalance = 1), 0)  
 FROM CustomerDetails.Customers AS c  
 JOIN INSERTED AS i  
 ON i.CustomerId = c.CustomerId  
END  
GO

*Листинг 47 – Теперь нулевое значение возвращает 0*

### Создание триггера DDL.

Был создан DDL триггер, который выполняется при создании хранимой процедуры, её изменении или удалении.

CREATE TRIGGER trgSprocs  
ON DATABASE  
FOR CREATE\_PROCEDURE, ALTER\_PROCEDURE, DROP\_PROCEDURE  
AS  
IF *DATEPART*(hh, *GETDATE*()) > 9 AND *DATEPART*(hh, *GETDATE*()) < 17  
BEGIN  
 DECLARE @Message nvarchar(max)  
 SELECT @Message = 'Completing work during core hours. Trying to release -' +  
 *EVENTDATA*().*value* ('(/EVENT\_INSTANCE/TSQLCommand/CommandText)[1]', 'nvarchar(max)')  
 RAISERROR (@Message, 16, 1)  
 ROLLBACK  
 EXEC msdb.dbo.sp\_send\_dbmail  
 @profile\_name = 'SQL Server Database Mail Profile',  
 @recipients = 'exam@limtu.spb.ru', @body = 'A stored procedure change',  
 @subject = 'A stored procedure change has been initiated and rolled back during core hours'  
END  
GO

*Листинг 48 – Код создания DDL триггера*

Работа триггера была проверена следующим кодом:

CREATE PROC *Test1* AS SELECT ‘Hello all’  
GO

*Листинг 49 – Проверка работы триггера*

Так как результат зависит от времени тестирования, триггер успешно был создан.



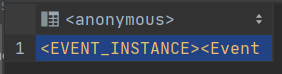
*Рисунок 106 – Процедура успешно создана*

Создадим триггер, который будет предоставлять информацию о любом действии в формате XML.

CREATE TRIGGER trgDBDump  
 ON DATABASE  
 FOR DDL\_DATABASE\_LEVEL\_EVENTS  
AS SELECT *EVENTDATA*()  
GO

*Листинг 50 – Создания триггера для любых действий над базой данных*

После создания тестовой процедуры был получен следующий результат:

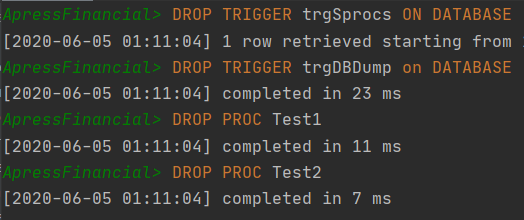


*Рисунок 107 – Данные о действии в формате XML*

### Удаление триггеров и тестовых процедур.

DROP TRIGGER trgSprocs ON DATABASE  
GO  
DROP TRIGGER trgDBDump on DATABASE  
GO  
  
DROP PROC Test1  
GO  
DROP PROC Test2  
GO

*Листинг 51 – Удаление триггеров и процедур*

**

*Рисунок 108 – Триггеры и процедуры успешно удалены*

## Окончательные результаты.

### Были созданы два вида триггеров. В триггеры вносились изменения. После всех тестов триггеры и тестовые процедуры были удалены.

## Выводы и анализ результатов работы.

В ходе данной лабораторной работы были рассмотрены два вида триггеров. Было рассмотрено создание триггеров, их редактирование. Созданные триггеры были протестированы на работоспособность, а после всех тестов были удалены.

# «Управление транзакциями и блокировками»

## Цель работы.

Изучить управление транзакциями и блокировками.

## Задачи, решаемые при выполнении работы.

### Определение транзакции.

### Поиск и обнаружение блокирования.

### Изучение уровней изоляции.

## Объект исследования.

Microsoft SQL Server, учебная база данных.

## Исходные данные.

### Методическое пособие.

### Учебная база данных.

## Выполнение работы.

### Создание транзакции с использованием инструкций COMMIT TRAN и ROLLBACK TRAN.

USE ApressFinancial  
GO  
  
SELECT 'Before', ShareId, [Description], CurrentPrice  
FROM ShareDetails.Shares  
WHERE ShareId = 3;  
  
BEGIN TRAN ShareUpd;  
 UPDATE ShareDetails.Shares  
 SET CurrentPrice = CurrentPrice \* 1.1  
 WHERE ShareId = 3;  
  
 SELECT 'During', ShareId, [Description], CurrentPrice  
 FROM ShareDetails.Shares  
 WHERE ShareId = 3;  
COMMIT TRAN;  
  
SELECT 'After', ShareId, [Description], CurrentPrice  
FROM ShareDetails.Shares  
WHERE ShareId = 3;  
GO

*Листинг 52 – Создание транзакции с использованием инструкции COMMIT TRAN*







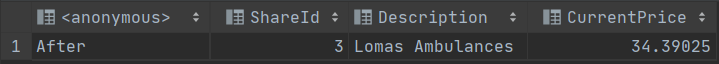
*Рисунок 109 – Результат работы транзакции*

USE ApressFinancial  
GO  
  
SELECT 'Before', ShareId, [Description], CurrentPrice  
FROM ShareDetails.Shares  
WHERE ShareId = 3;  
  
BEGIN TRAN ShareUpd;  
 UPDATE ShareDetails.Shares  
 SET CurrentPrice = CurrentPrice \* 1.1  
 WHERE ShareId = 3;  
  
 SELECT 'During', ShareId, [Description], CurrentPrice  
 FROM ShareDetails.Shares  
 WHERE ShareId = 3;  
ROLLBACK TRAN;  
  
SELECT 'After', ShareId, [Description], CurrentPrice  
FROM ShareDetails.Shares  
WHERE ShareId = 3;  
GO

*Листинг 53 – Создание транзакции с использованием инструкции COMMIT TRAN*

**

**

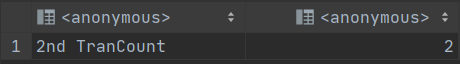
**

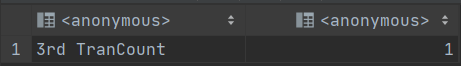
*Рисунок 110 – Результат работы процедуры*

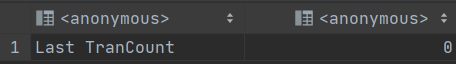
BEGIN TRAN ShareUpd  
 SELECT '1st TranCount', *@@TRANCOUNT*BEGIN TRAN ShareUpd2  
 SELECT '2nd TranCount', *@@TRANCOUNT*COMMIT TRAN ShareUpd2  
 SELECT '3rd TranCount', *@@TRANCOUNT*COMMIT TRAN  
 SELECT 'Last TranCount', *@@TRANCOUNT*GO

*Листинг 54 – Вложенные транзакции*









*Рисунок 111 – Результат работы вложенных транзакций*

### Поиск и обнаружение блокирования.

Был выполнен файл SQLQuery\_CREATE\_DB\_exmp\_block.sql, с помощью которого была создана новая база данных [test\_block].

Было сделано три подключения к базе данных. В первом подключении был выполнен следующий код:

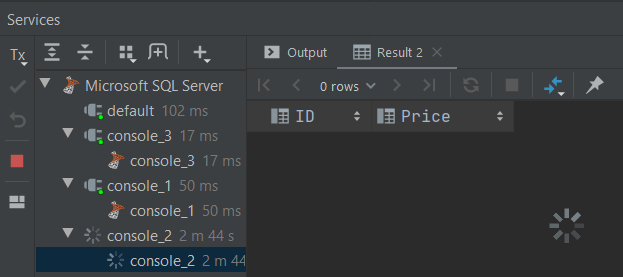
USE [test\_block]  
GO  
  
BEGIN TRAN;  
 UPDATE dbo.[t1]  
 SET [Price] = [Price] + 1.00  
 WHERE [ID] = 2;  
SELECT [ID], [Price] FROM dbo.[t1] WHERE [ID] = 2;

*Листинг 55 – Выполнение транзакции без его завершения*

Во втором подключении был выполнен следующий код:

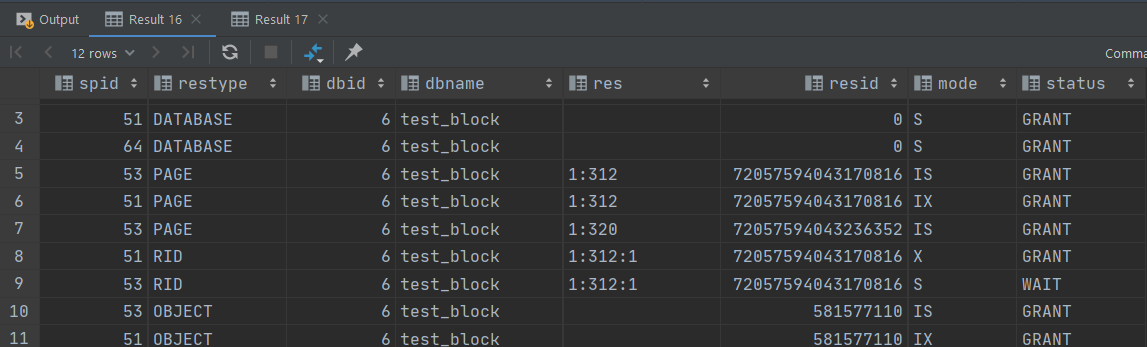
USE [test\_block]  
GO  
  
SELECT [ID], [Price] FROM dbo.[t1] WHERE [ID] = 2;

*Листинг 56 – Запрос к строке, которая участвовала в незавершённой транзакции*

**

*Рисунок 112 – Второй подключение вынуждено ждать, тогда как первое уже отработало*

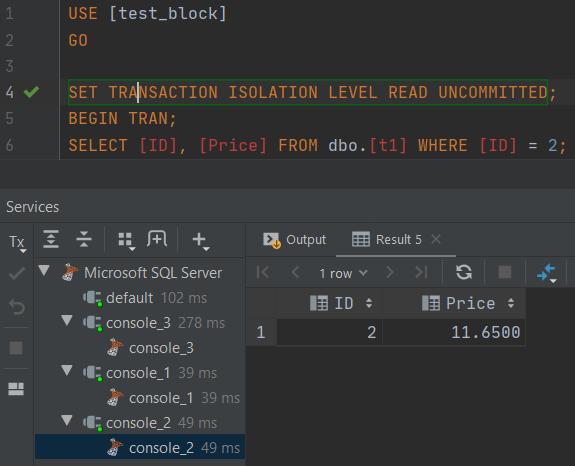
В третьем соединении с базой данных были рассмотрены различные представления данных о процессах.



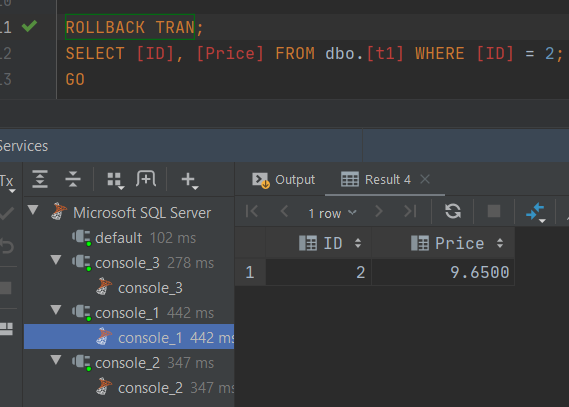
*Рисунок 113 – Данные о текущих процессах*

### Уровни изоляции.

Был выполнен код из прошлого задания, который заблокировал запись с ID = 2. Однако, во втором подключении был задан уровень изоляции READ UNCOMMITED. Код успешно выполнился.



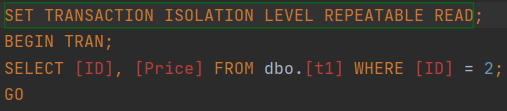
*Рисунок 114 – Код успешно выполнен*

**

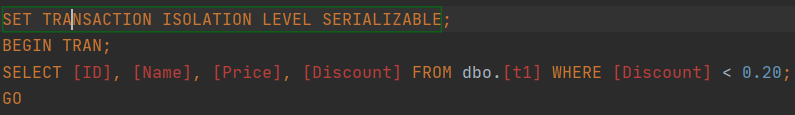
*Рисунок 115 – Откат выполнения транзакции*

Вновь выполним код из прошлого задания, чтобы заблокировать поле. Во втором подключении зададим уровень изоляции READ COMMITED. После выполнения кода во втором подключении результат не был получен. Так как этот уровень изоляции подразумевает, что транзакцию необходимо завершить. Отменим транзакцию с помощью кода в первом подключении.

Были рассмотрены оставшиеся уровни изоляции:



*Рисунок 116 – Уровень изоляции REPETABLE READ*

**

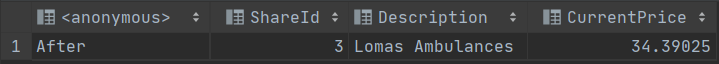
*Рисунок 117 – Уровень изоляции SERIALIZABLE*

## Окончательные результаты.

### Были созданы транзакции, рассмотрена работа вложенных транзакций.

**

**

**

*Рисунок 118 – Результат работы процедуры*

### На основе созданных транзакций были рассмотрены блокировки и уровни изоляции.

## Выводы и анализ результатов работы.

Выполнена заключительная лабораторная работа по проектированию и реализации баз данных. В данной работе были изучены транзакции, способы их создания, инструкции COMMIT TRAN и ROLLBACK TRAN. Изучено, как получать сведения о блокировках, как задавать уровни изоляции транзакций. Рассмотрены уровни READ UNCOMMITED, READ COMMITED, REPEATABLE READ, SERIALIZABLE и как они работают.