[Теоретическая часть 4](#_Toc135776916)

[База данных 4](#_Toc135776917)

[Реляционные 4](#_Toc135776918)

[SQL 4](#_Toc135776919)

[Целостность данных 5](#_Toc135776920)

[Транзакции 5](#_Toc135776921)

[Соответствие требованиям ACID 6](#_Toc135776922)

[Шифрование 6](#_Toc135776923)

[Шифрование SQL Server 7](#_Toc135776924)

[Функции Transact-SQL 8](#_Toc135776925)

[Асимметричные ключи 8](#_Toc135776926)

[Симметричные ключи 9](#_Toc135776927)

[Сертификаты 9](#_Toc135776928)

[Прозрачное шифрование данных (TDE) 10](#_Toc135776929)

[Выбор алгоритма шифрования 11](#_Toc135776930)

[Ключи шифрования базы данных 11](#_Toc135776931)

[Главный ключ службы (SMK) 12](#_Toc135776932)

[Главный ключ базы данных (DMK) 12](#_Toc135776933)

[Управление SQL Server и ключами базы данных 13](#_Toc135776934)

[Практическая часть 14](#_Toc135776935)

[Создание базы данных 14](#_Toc135776936)

[Создание таблиц 15](#_Toc135776937)

[Создании таблицы Товар 15](#_Toc135776938)

[Создание таблицы Позиция Заказа 16](#_Toc135776939)

[Создание таблицы Заказ 17](#_Toc135776940)

[Создание таблицы Клиент 18](#_Toc135776941)

[Создание таблицы Сотрудник 19](#_Toc135776942)

[Создание таблицы Должность 20](#_Toc135776943)

[Создание таблицы Статус Заказа 21](#_Toc135776944)

[Создание связей между таблицами 22](#_Toc135776945)

[Добавление данных в таблицы 24](#_Toc135776946)

[Заполнение таблицы заказов 24](#_Toc135776947)

[Заполнение таблицы клиентов 25](#_Toc135776948)

[Заполнение таблицы позиций заказов 25](#_Toc135776949)

[Заполнение таблицы статусов заказов 26](#_Toc135776950)

[Заполнение таблицы должностей 26](#_Toc135776951)

[Заполнение таблицы товаров 27](#_Toc135776952)

[Заполнение таблицы сотрудников 27](#_Toc135776953)

[Создание шифрования при помощи средств MsSql 29](#_Toc135776954)

[Создание мастер ключа 29](#_Toc135776955)

[Создание сертификата 30](#_Toc135776956)

[Резервное копирование зашифрованной базы данных 31](#_Toc135776957)

[Хеширование данных с помощью возможностей C# 33](#_Toc135776958)

[Создание базы данных 33](#_Toc135776959)

[Создание приложения WPF 34](#_Toc135776960)

Теоретическая часть

База данных

База данных — это хранилище, куда приложение складывает свои данные. Если приложение небольшое, отдельная база не нужна. Но потом это становится удобнее и выгоднее с точки зрения памяти. Базы данных в основном делятся на реляционные и не реляционные.

Реляционные

Реляционная база данных – это набор данных с предопределенными связями между ними. Эти данные организованны в виде набора таблиц, состоящих из столбцов и строк. В таблицах хранится информация об объектах, представленных в базе данных. В каждом столбце таблицы хранится определенный тип данных, в каждой ячейке – значение атрибута. Каждая стока таблицы представляет собой набор связанных значений, относящихся к одному объекту или сущности. Каждая строка в таблице может быть помечена уникальным идентификатором, называемым первичным ключом, а строки из нескольких таблиц могут быть связаны с помощью внешних ключей. К этим данным можно получить доступ многими способами, и при этом реорганизовывать таблицы БД не требуется. Главным представителем реляционных баз данных является SQL.



SQL

SQL — это язык программирования, предназначенный для работы с наборами фактов и отношениями между ними. В программах управления реляционными базами данных, таких как Microsoft Office Access, язык SQL используется для работы с данными. В отличие от многих языков программирования, SQL удобочитаем и понятен даже новичкам. Как и многие языки программирования, SQL является международным стандартом, признанным такими комитетами по стандартизации, как [ISO](https://www.iso.org/) и [ANSI](https://www.ansi.org/).

На языке SQL описываются наборы данных, помогающие получать ответы на вопросы. При использовании SQL необходимо применять правильный синтаксис. Синтаксис — это набор правил, позволяющих правильно сочетать элементы языка. Синтаксис SQL основан на синтаксисе английского языка и имеет много общих элементов с синтаксисом языка Visual Basic для приложений (VBA).

Например, простая инструкция SQL, извлекающая список фамилий контактов с именем Mary, может выглядеть следующим образом:

SELECT Last\_Name  
FROM Contacts  
WHERE First\_Name = 'Mary';

Целостность данных

Целостность данных – это полнота, точность и единообразие данных. Для поддержания целостности данных в реляционных БД используется ряд инструментов. В их число входят первичные ключи, внешние ключи, ограничения «Not NULL», «Unique», «Default» и «Check». Эти ограничения целостности позволяют применять практические правила к данным в таблицах и гарантировать точность и надежность данных. Большинство ядер БД также поддерживает интеграцию пользовательского кода, который выполняется в ответ на определенные операции в БД.

Транзакции

Транзакция в базе данных – это один или несколько операторов SQL, выполненных в виде последовательности операций, представляющих собой единую логическую задачу. Транзакция представляет собой неделимое действие, то есть она должна быть выполнена как единое целое и либо должна быть записана в базу данных целиком, либо не должен быть записан ни один из ее компонентов. В терминологии реляционных баз данных транзакция завершается либо действием COMMIT, либо ROLLBACK. Каждая транзакция рассматривается как внутренне связный, надежный и независимый от других транзакций элемент.

Соответствие требованиям ACID

Для соблюдения целостности данных все транзакции в БД должны соответствовать требованиям ACID, то есть быть атомарными, единообразными, изолированными и надежными.

**Атомарность** – это условие, при котором либо транзакция успешно выполняется целиком, либо, если какая-либо из ее частей не выполняется, вся транзакция отменяется. **Единообразие** – это условие, при котором данные, записываемые в базу данных в рамках транзакции, должны соответствовать всем правилам и ограничениям, включая ограничения целостности, каскады и триггеры. **Изолированность** необходима для контроля над согласованностью и гарантирует базовую независимость каждой транзакции. **Надежность** подразумевает, что все внесенные в базу данных изменения на момент успешного завершения транзакции считаются постоянными.

Шифрование

**Шифрова́ние** — обратимое преобразование [информации](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%86%D0%B8%D1%8F) в целях сокрытия от [неавторизованных](https://ru.wikipedia.org/wiki/%D0%90%D0%B2%D1%82%D0%BE%D1%80%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D1%8F) лиц с предоставлением в это же время [авторизованным](https://ru.wikipedia.org/wiki/%D0%90%D0%B2%D1%82%D0%BE%D1%80%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D1%8F) пользователям доступа к ней. Главным образом, шифрование служит для соблюдения [конфиденциальности](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BD%D1%84%D0%B8%D0%B4%D0%B5%D0%BD%D1%86%D0%B8%D0%B0%D0%BB%D1%8C%D0%BD%D0%BE%D1%81%D1%82%D1%8C) передаваемой информации. Важной особенностью любого алгоритма шифрования является использование [ключа](https://ru.wikipedia.org/wiki/%D0%9A%D0%BB%D1%8E%D1%87_(%D0%BA%D1%80%D0%B8%D0%BF%D1%82%D0%BE%D0%B3%D1%80%D0%B0%D1%84%D0%B8%D1%8F)), который утверждает выбор конкретного преобразования из совокупности возможных для данного алгоритма.

Пользователи являются *авторизованными*, если они обладают определённым [аутентичным](https://ru.wikipedia.org/wiki/%D0%90%D1%83%D1%82%D0%B5%D0%BD%D1%82%D0%B8%D1%84%D0%B8%D0%BA%D0%B0%D1%86%D0%B8%D1%8F) [ключом](https://ru.wikipedia.org/wiki/%D0%9A%D0%BB%D1%8E%D1%87_(%D0%BA%D1%80%D0%B8%D0%BF%D1%82%D0%BE%D0%B3%D1%80%D0%B0%D1%84%D0%B8%D1%8F)). Вся сложность и, собственно, задача шифрования состоит в том, как именно реализован этот процесс.

В целом, шифрование состоит из двух составляющих: зашифрование и расшифрование.

С помощью шифрования обеспечиваются три состояния безопасности информации:

* Конфиденциальность.
* Шифрование используется для скрытия информации от неавторизованных пользователей при передаче или при хранении.
* Целостность.
* Шифрование используется для предотвращения изменения информации при передаче или хранении.
* Идентифицируемость.
* Шифрование используется для аутентификации источника информации и предотвращения отказа отправителя информации от того факта, что данные были отправлены именно им.

Для того, чтобы прочитать зашифрованную информацию, принимающей стороне необходимы [ключ](https://ru.wikipedia.org/wiki/%D0%9A%D0%BB%D1%8E%D1%87_(%D0%BA%D1%80%D0%B8%D0%BF%D1%82%D0%BE%D0%B3%D1%80%D0%B0%D1%84%D0%B8%D1%8F)) и [дешифратор](https://ru.wikipedia.org/wiki/%D0%94%D0%B5%D1%88%D0%B8%D1%84%D1%80%D0%B0%D1%82%D0%BE%D1%80) (устройство, реализующее алгоритм расшифровывания). Идея шифрования состоит в том, что злоумышленник, перехватив зашифрованные данные и не имея к ним ключа, не может ни прочитать, ни изменить передаваемую информацию. Кроме того, в современных [криптосистемах (с открытым ключом)](https://ru.wikipedia.org/wiki/%D0%9A%D1%80%D0%B8%D0%BF%D1%82%D0%BE%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0_%D1%81_%D0%BE%D1%82%D0%BA%D1%80%D1%8B%D1%82%D1%8B%D0%BC_%D0%BA%D0%BB%D1%8E%D1%87%D0%BE%D0%BC) для шифрования, расшифрования данных могут использоваться разные ключи. Однако, с развитием криптоанализа, появились методики, позволяющие дешифровать закрытый текст без ключа. Они основаны на математическом анализе переданных данных.

Шифрование SQL Server

Шифрование представляет собой способ скрытия данных с помощью ключа или пароля. Это делает данные бесполезными без соответствующего ключа или пароля для дешифрования. Шифрование не решает проблемы управления доступом. Однако оно повышает защиту за счет ограничения потери данных даже при обходе системы управления доступом. Например, если компьютер, на котором установлена база данных, был настроен не правильно и злоумышленник смог получить конфиденциальные данные, то украденная информация будет бесполезна, если она была предварительно зашифрована.

## Иерархия шифрования

[SQL Server](https://ru.wikipedia.org/wiki/Microsoft_SQL_Server) шифрует данные с помощью иерархической инфраструктуры шифрования и управления [ключами](https://ru.wikipedia.org/wiki/%D0%9A%D0%BB%D1%8E%D1%87_(%D0%BA%D1%80%D0%B8%D0%BF%D1%82%D0%BE%D0%B3%D1%80%D0%B0%D1%84%D0%B8%D1%8F)). Каждый слой шифрует слой под ним с помощью комбинации сертификатов, асимметричных ключей и симметричных ключей. Асимметричные и симметричные ключи могут храниться за пределами SQL Server в модуле расширенного управления [ключами](https://ru.wikipedia.org/wiki/%D0%9A%D0%BB%D1%8E%D1%87_(%D0%BA%D1%80%D0%B8%D0%BF%D1%82%D0%BE%D0%B3%D1%80%D0%B0%D1%84%D0%B8%D1%8F)) (EKM).

Каждый уровень иерархии шифрования шифрует слой под ним и отображает наиболее распространенные конфигурации шифрования. Доступ к началу иерархии обычно защищен паролем.

Так же стоит учитывать следующие принципы:

* Для лучшей производительности шифруйте данные с помощью симметричных ключей вместо сертификатов или асимметричных ключей.
* Главные ключи базы данных защищены главным ключом службы. Главный ключ службы создается программой установки SQL Server и шифруется с помощью API защиты данных Windows ([DPAPI](https://ru.wikipedia.org/wiki/Data_Protection_API))
* Возможны и другие иерархии шифрования, устанавливающие дополнительные слои.
* Модуль Extensible Key Management (EKM) содержит симметричные или асимметричные ключи вне SQL Server.
* Прозрачное шифрование данных (TDE) должно использовать симметричный ключ, называемый ключом шифрования базы данных, который защищен либо сертификатом, защищенным главным ключом базы данных master database, либо асимметричным ключом, хранящимся в EKM.
* Главный ключ службы и все главные ключи базы данных являются симметричными ключами.

SQL Server предоставляет следующие механизмы шифрования:

* [Функции Transact-SQL](https://ru.wikipedia.org/wiki/Transact-SQL)
* [Асимметричные ключи](https://ru.wikipedia.org/wiki/%D0%9A%D1%80%D0%B8%D0%BF%D1%82%D0%BE%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0_%D1%81_%D0%BE%D1%82%D0%BA%D1%80%D1%8B%D1%82%D1%8B%D0%BC_%D0%BA%D0%BB%D1%8E%D1%87%D0%BE%D0%BC)
* [Симметричные ключи](https://ru.wikipedia.org/wiki/%D0%A1%D0%B8%D0%BC%D0%BC%D0%B5%D1%82%D1%80%D0%B8%D1%87%D0%BD%D1%8B%D0%B5_%D0%BA%D1%80%D0%B8%D0%BF%D1%82%D0%BE%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D1%8B)
* Сертификаты
* Прозрачное шифрование данных (TDE)

[Функции Transact-SQL](https://ru.wikipedia.org/wiki/Transact-SQL)

Отдельные элементы могут быть зашифрованы при вставке или обновлении с использованием функций Transact-SQL.

[Асимметричные ключи](https://ru.wikipedia.org/wiki/%D0%9A%D1%80%D0%B8%D0%BF%D1%82%D0%BE%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0_%D1%81_%D0%BE%D1%82%D0%BA%D1%80%D1%8B%D1%82%D1%8B%D0%BC_%D0%BA%D0%BB%D1%8E%D1%87%D0%BE%D0%BC)

Асимметричный ключ состоит из закрытого ключа и соответствующего открытого ключа. Каждый ключ может расшифровать данные, зашифрованные другим. Асимметричное шифрование и дешифрование относительно ресурсоемки, но они обеспечивают более высокий уровень безопасности, чем симметричное шифрование. Асимметричный ключ может использоваться для шифрования симметричного ключа для хранения в базе данных.

[Симметричные ключи](https://ru.wikipedia.org/wiki/%D0%A1%D0%B8%D0%BC%D0%BC%D0%B5%D1%82%D1%80%D0%B8%D1%87%D0%BD%D1%8B%D0%B5_%D0%BA%D1%80%D0%B8%D0%BF%D1%82%D0%BE%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D1%8B)

Симметричный ключ-это один ключ, который используется как для шифрования, так и для дешифрования. Шифрование и дешифрование с помощью симметричного ключа выполняется быстро и подходит для повседневного использования с конфиденциальными данными в базе данных.

Сертификаты

Сертификат открытого ключа, обычно называемый сертификатом, является оператором с [цифровой подписью](https://ru.wikipedia.org/wiki/%D0%AD%D0%BB%D0%B5%D0%BA%D1%82%D1%80%D0%BE%D0%BD%D0%BD%D0%B0%D1%8F_%D0%BF%D0%BE%D0%B4%D0%BF%D0%B8%D1%81%D1%8C), который связывает значение открытого ключа с личностью человека, устройства или службы, которая содержит соответствующий закрытый ключ. Сертификаты выдаются и подписываются центром сертификации (ЦС). Объект, который получает сертификат от ЦС, является субъектом этого сертификата. Как правило, сертификаты содержат следующую информацию:

* Открытый ключ субъекта
* Идентификационные данные субъекта, такие как имя и адрес электронной почты
* Время, в течение которого сертификат считается действительным
* Информация об [эмитенте](https://ru.wikipedia.org/wiki/%D0%AD%D0%BC%D0%B8%D1%82%D0%B5%D0%BD%D1%82).
* Цифровая подпись эмитента.

Основное преимущество сертификатов заключается в том, что они освобождают [хосты](https://ru.wikipedia.org/wiki/%D0%A5%D0%BE%D1%81%D1%82) от необходимости поддерживать набор паролей для отдельных субъектов. Вместо этого хост просто устанавливает доверие к эмитенту сертификата, который затем может подписать неограниченное количество сертификатов. Когда хост, такой как защищенный веб-сервер, назначает эмитента доверенным корневым органом, хост неявно доверяет политикам, которые эмитент использовал для установления привязок сертификатов, которые он выдает. По сути, хост доверяет тому, что эмитент проверил идентичность субъекта сертификата. Хост назначает эмитента в качестве доверенного корневого центра, помещая самозаверяющий сертификат эмитента, который содержит открытый ключ эмитента, в хранилище сертификатов доверенного корневого центра сертификации на главном компьютере. Промежуточные или подчиненные центры сертификации являются доверенными, только если они имеют действительный путь сертификации от доверенного корневого центра сертификации. Самозаверяющие сертификаты, созданные SQL Server, соответствуют [стандарту X.509](https://ru.wikipedia.org/wiki/X.509) и поддерживают поля X.509 v1.

Прозрачное шифрование данных (TDE)

Прозрачное шифрование данных (TDE) шифрует файлы данных [SQL Server](https://ru.wikipedia.org/wiki/Microsoft_SQL_Server), базы данных [SQL Azure](https://ru.wikipedia.org/wiki/SQL_Azure) и Azure Synapse Analytics (SQL DW), известные как шифрование данных в состоянии покоя. Для обеспечения безопасности базы данных можно принять ряд мер предосторожности, таких как разработка защищенной системы, шифрование конфиденциальных ресурсов и создание брандмауэра вокруг серверов баз данных. Однако в случае кражи физического носителя злоумышленник может просто восстановить или присоединить базу данных и просмотреть её. Одним из решений является шифрование конфиденциальных данных в базе и защита ключей, используемых для шифрования с помощью сертификата. Это не позволяет никому без ключей использовать данные, но этот вид защиты должен быть запланирован заранее.

#### **Реализация Microsoft**

TDE выполняет шифрование и расшифровку данных и файлов журналов в режиме реального времени. При шифровании используется ключ шифрования базы данных (DEK), который хранится в загрузочной записи базы данных для обеспечения доступности во время восстановления. DEK — это симметричный ключ, защищенный с помощью сертификата, хранящегося в главной базе данных сервера, или асимметричного ключа, защищенного модулем EKM. TDE защищает данные «в состоянии покоя», то есть данные и файлы журналов. Это дает возможность соблюдать многие законы, правила и руководящие принципы, установленные в различных отраслях промышленности. Так же позволяет разработчикам ПО шифровать данные с использованием алгоритмов шифрования [AES](https://ru.wikipedia.org/wiki/Advanced_Encryption_Standard) и [3DES](https://ru.wikipedia.org/wiki/Triple_DES) без изменения существующих приложений.

#### **Реализация Oracle**

TDE применяется для файлов БД на уровне столбцов. Для таблицы, содержащей выбранные к шифрованию столбцы, создаётся симметричный ключ шифрования, защищённый главным ключом, который хранится в безопасном месте за пределами БД, называемом бумажником ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *Wallet*). Зашифрованные ключи таблиц содержатся в словаре данных ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *Data Dictionary*).

Выбор алгоритма шифрования

Шифрование является одним из нескольких способов защиты, доступных администратору. Алгоритмы шифрования определяют преобразования данных, которые не могут быть легко отменены неавторизованными пользователями. SQL Server позволяет администраторам и разработчикам выбирать один из нескольких алгоритмов, включая [DES](https://ru.wikipedia.org/wiki/DES), [Triple DES](https://ru.wikipedia.org/wiki/Triple_DES), TRIPLE\_DES\_3KEY, [RC2](https://ru.wikipedia.org/wiki/RC2), [RC4](https://ru.wikipedia.org/wiki/RC4), 128-bit RC4, [DESX](https://ru.wikipedia.org/wiki/DESX), 128-bit [AES](https://ru.wikipedia.org/wiki/AES), 192-bit AES и 256-bit AES. При выборе алгоритма шифрования следует учитывать следующие принципы:

* сильное шифрование обычно потребляет больше ресурсов процессора, чем слабое
* длинные ключи обычно обеспечивают более надежное шифрование, чем короткие
* асимметричное шифрование работает медленнее, чем симметричное
* длинные, сложные пароли сильнее коротких
* симметричное шифрование обычно рекомендуется, когда ключ хранится только локально, асимметричное шифрование рекомендуется, когда ключи должны передаваться по сети
* если вы шифруете много данных, вы должны зашифровать данные с помощью симметричного ключа и зашифровать симметричный ключ с помощью асимметричного ключа
* зашифрованные данные не могут быть сжаты, но сжатые данные могут быть зашифрованы. Если вы используете сжатие, вы должны сжать данные перед их шифрованием

Разъяснения относительно алгоритмов DES: DESX был неправильно назван. Симметричные ключи, созданные с помощью ALGORITHM = DESX, фактически используют шифр TRIPLE DES с 192-битным ключом. Алгоритм DESX не предусмотрен. Эта функция находится в режиме обслуживания и может быть удалена в будущей версии Microsoft SQL Server. Избегайте использования этой функции в новых разработках и планируйте модифицировать приложения, которые в настоящее время используют эту функцию. Симметричные ключи, созданные с помощью ALGORITHM = TRIPLE\_DES\_3KEY, используют TRIPLE DES с 192-битным ключом. Симметричные ключи, созданные с ALGORITHM = TRIPLE\_DES, используют TRIPLE DES с 128-битным ключом.

Ключи шифрования базы данных

В SQL Server ключи шифрования включают в себя сочетание открытых, закрытых и симметричных ключей, которые используются для защиты конфиденциальных данных. Симметричный ключ создается во время инициализации SQL Server при первом запуске экземпляра SQL Server. Ключ используется SQL Server для шифрования конфиденциальных данных, хранящихся в SQL Server. Открытый и закрытый ключи создаются операционной системой и используются для защиты симметричного ключа. Пара открытого и закрытого ключей создается для каждого экземпляра SQL Server, который хранит конфиденциальные данные в базе данных. SQL Server имеет два основных приложения для ключей: главный ключ службы (SMK), созданный на экземпляре SQL Server и для него, и главный ключ базы данных (DMK), используемый для базы данных.

Главный ключ службы (SMK)

Главный ключ службы является корнем иерархии шифрования SQL Server. SMK генерируется автоматически при первом запуске экземпляра SQL Server и используется для шифрования пароля связанного сервера, учётных данных и главного ключа базы данных. SMK шифруется с помощью ключа локального компьютера с помощью API защиты данных Windows ([DPAPI](https://ru.wikipedia.org/wiki/Data_Protection_API)). DPAPI использует ключ, полученный из учётных данных Windows, учётной записи службы SQL Server и учётных данных компьютера. Главный ключ службы может быть расшифрован только учётной записью службы, под которой он был создан, или участником, имеющим доступ к учётным данным компьютера. Главный ключ службы может быть открыт только той учётной записью службы Windows, под которой он был создан, или принципалом, имеющим доступ как к имени учётной записи службы, так и к её паролю. SQL Server использует алгоритм шифрования [AES](https://ru.wikipedia.org/wiki/Advanced_Encryption_Standard) для защиты главного ключа службы (SMK) и главного ключа базы данных (DMK). AES — более новый алгоритм шифрования, чем [3DES](https://ru.wikipedia.org/wiki/Triple_DES), используемый в более ранних версиях.

Главный ключ базы данных (DMK)

Главный ключ базы данных — это симметричный ключ, который используется для защиты закрытых ключей сертификатов и асимметричных ключей, присутствующих в базе данных. Он также может использоваться для шифрования данных, но имеет ограничения по длине, которые делают его менее практичным для данных, чем использование симметричного ключа. Чтобы включить автоматическое дешифрование главного ключа базы данных, копия ключа шифруется с помощью SMK. Он хранится как в базе данных, в которой он используется, так и в базе данных главной системы. Копия DMK, хранящаяся в базе данных главной системы, автоматически обновляется при каждом изменении DMK. Однако это значение по умолчанию можно изменить с помощью параметра DROP ENCRYPTION BY SERVICE MASTER KEY в инструкции ALTER MASTER KEY. DMK, который не зашифрован главным ключом службы, должен быть открыт с помощью оператора OPEN MASTER KEY и пароля.

Управление SQL Server и ключами базы данных

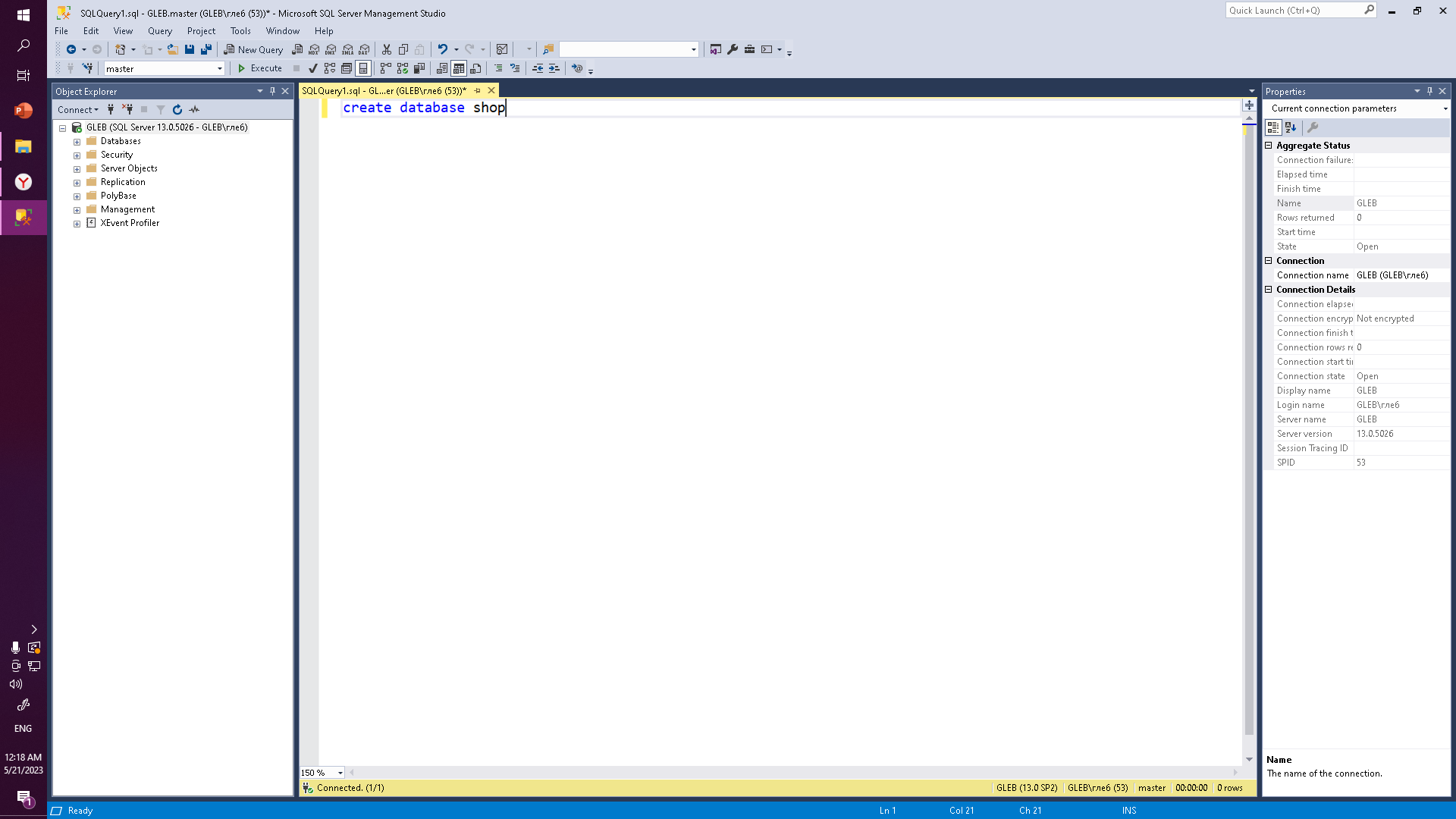
### Управление ключами шифрования состоит из создания новых ключей базы данных, создания резервной копии ключей сервера и базы данных, а также определения того, когда и как восстановить, удалить или изменить ключи. Для управления симметричными ключами можно использовать средства, входящие в состав SQL Server, чтобы выполнить следующие действия:

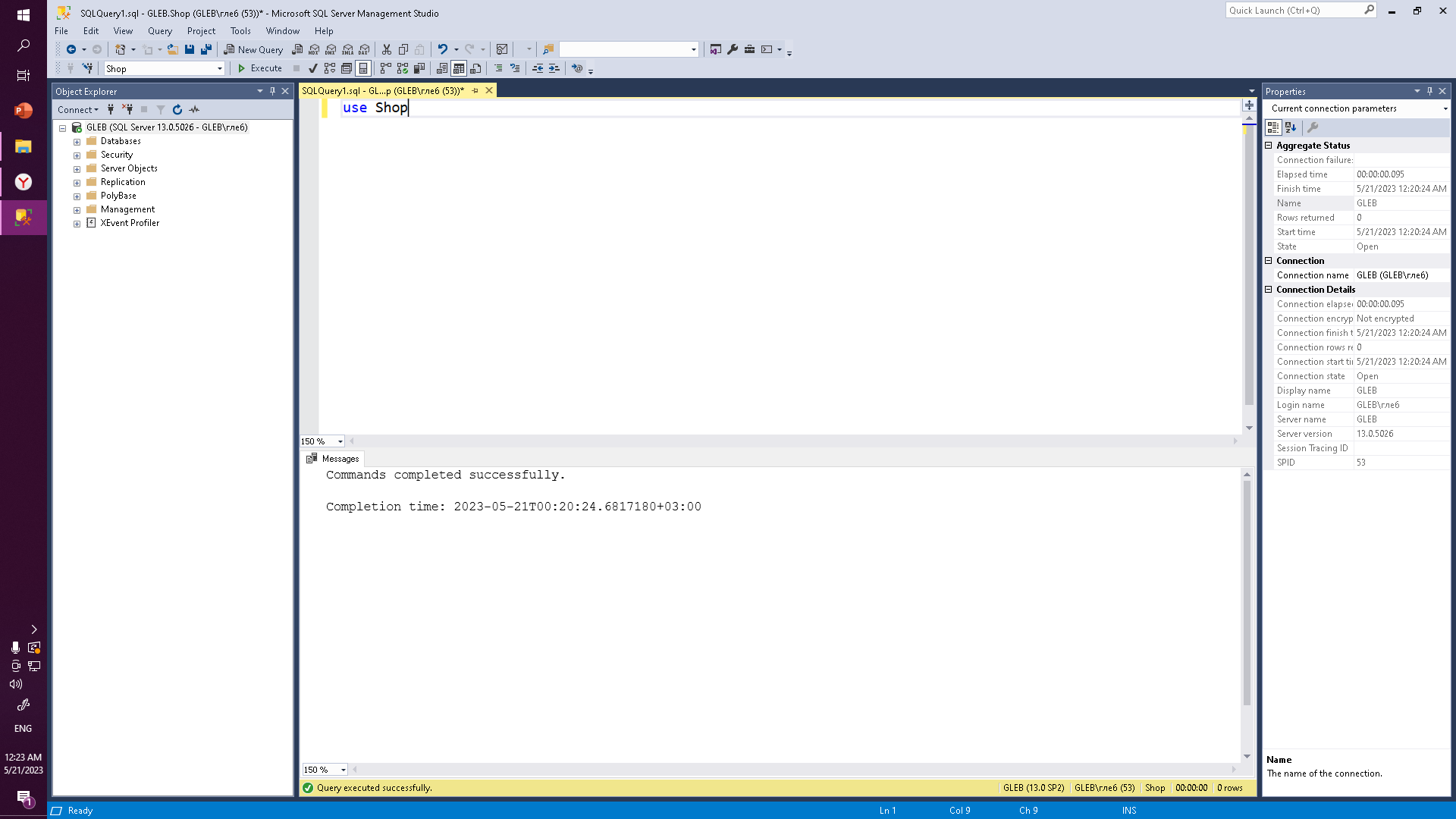
* создайте резервную копию ключей сервера и базы данных, чтобы использовать их для восстановления установки сервера или в рамках запланированной миграции
* восстановите ранее сохраненный ключ в базе данных. Это позволяет новому экземпляру сервера получить доступ к существующим данным, которые он изначально не шифровал
* удалите зашифрованные данные в базе, когда вы больше не можете получить доступ к зашифрованным данным.
* повторно создайте ключи и повторно зашифруйте данные, если ключ будет скомпрометирован. В целях безопасности рекомендуется периодически (например, каждые несколько месяцев) создавать ключи заново, чтобы защитить сервер от атак, которые пытаются расшифровать ключи
* добавление или удаление экземпляра сервера из масштабного развертывания, в котором несколько серверов совместно используют одну базу данных и ключ, обеспечивающий обратимое шифрование для этой базы данных.

Практическая часть

Создание базы данных

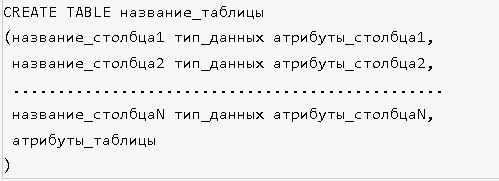
Для того, чтобы создать базу данных необходимо воспользоваться командой CREATE DATABASE “Название базы данных”



Для создания и изменения таблиц в данной базе данных прописываем команду USE “Название базы данных”

Создание таблиц

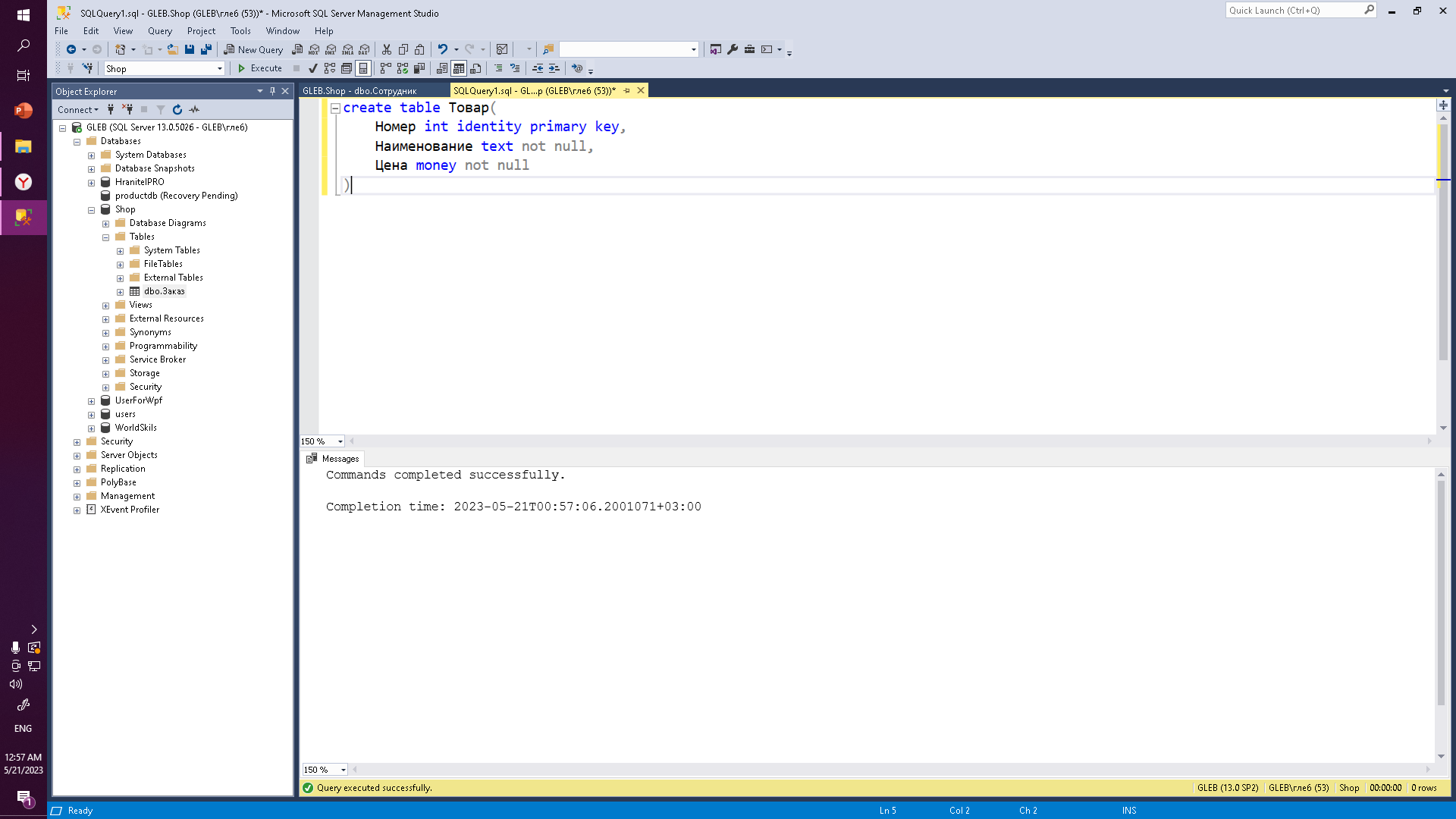
После всех подготовительных процедур можно переходить к созданию таблиц. Для создания таблиц будет использоваться команда create table, имеющая такой синтаксис:



Для добавления масок используется конструкция alter table в сочетании с check и регулярными выражениями

Создании таблицы Товар

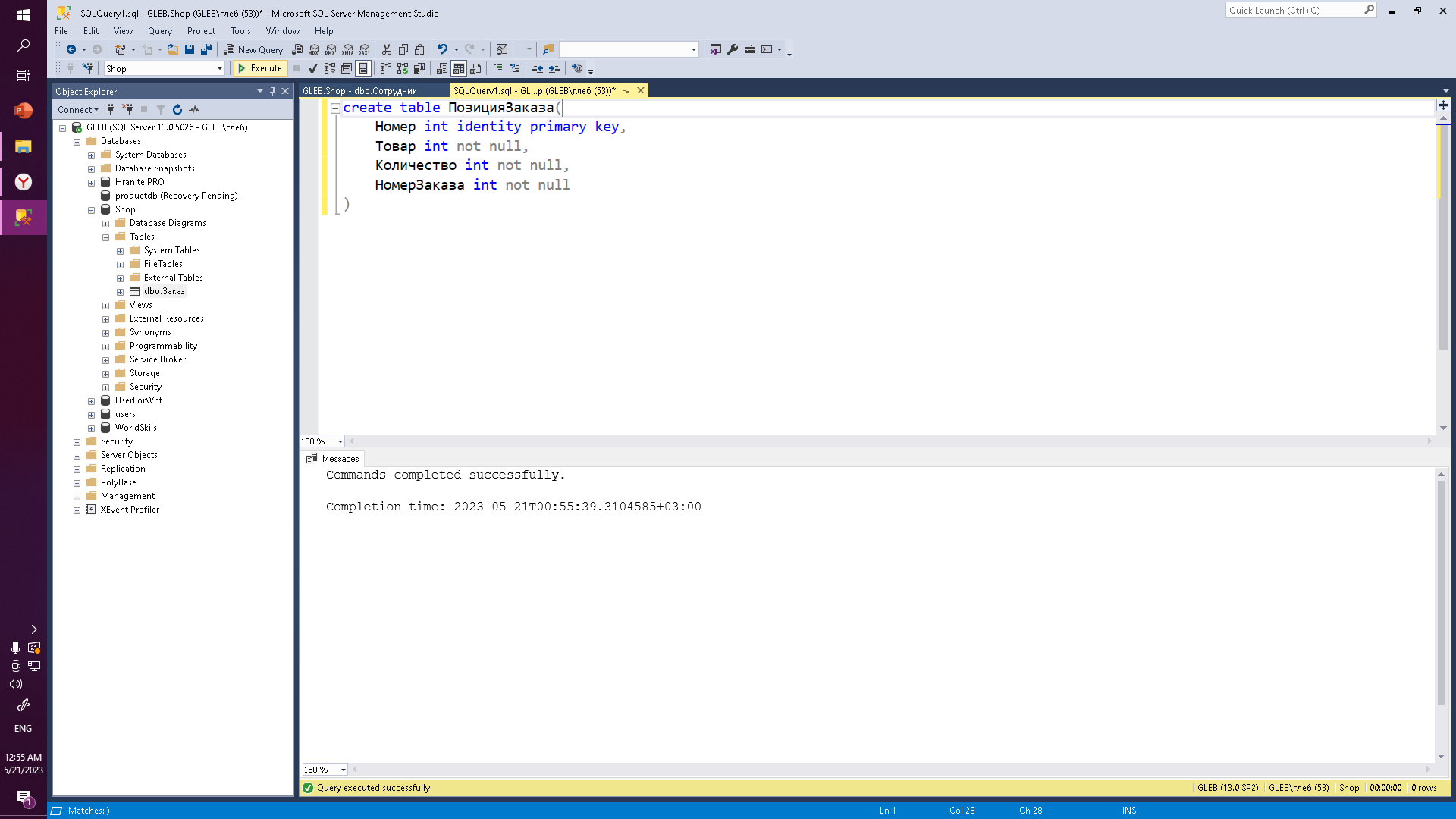
Первой таблицей будет таблица товаров, которая будет содержать 3 поля (Номер, Наименование, Цена)



Поле номер будет иметь целочисленный тип для того, чтобы соединить таблицу с последующими. Наименование имеет текстовый тип, так как название товара в большинстве случае это текст. Цена соответственно использует денежный тип

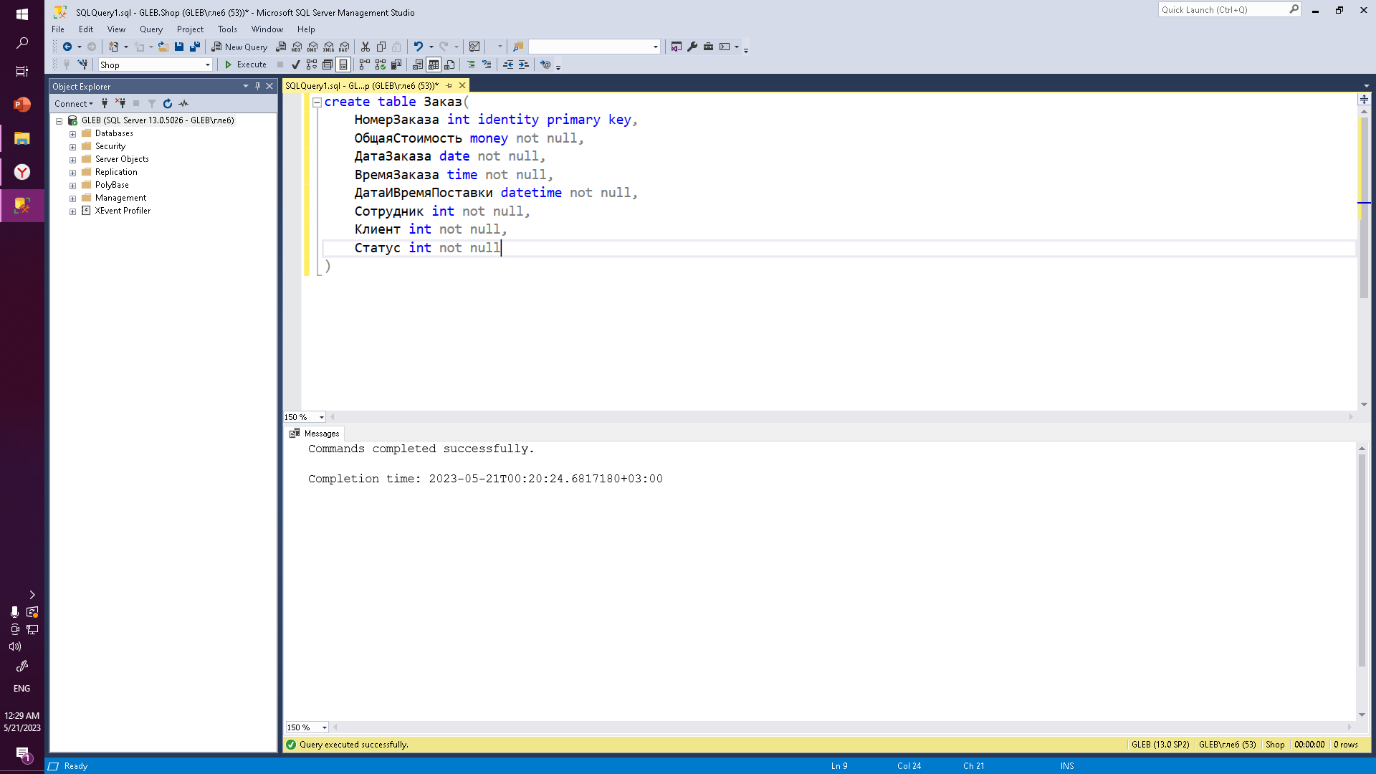
Создание таблицы Позиция Заказа

Перейдем к созданию таблицы Позиция Заказа, имеющей четыре поля (Номер, Товар, Количество, Номер Заказа)



Поле номер будет иметь целочисленный тип для того, чтобы соединить таблицу с последующими. Поле товар тоже имеет целочисленный тип, чтобы по нему соединить эту таблицу с таблицей” Товар ”. Поле количество имеет целочисленный тип для учета количества купленных товаров. Поле номер заказа с целочисленным типом необходимо для соединения с таблицей “Заказ”

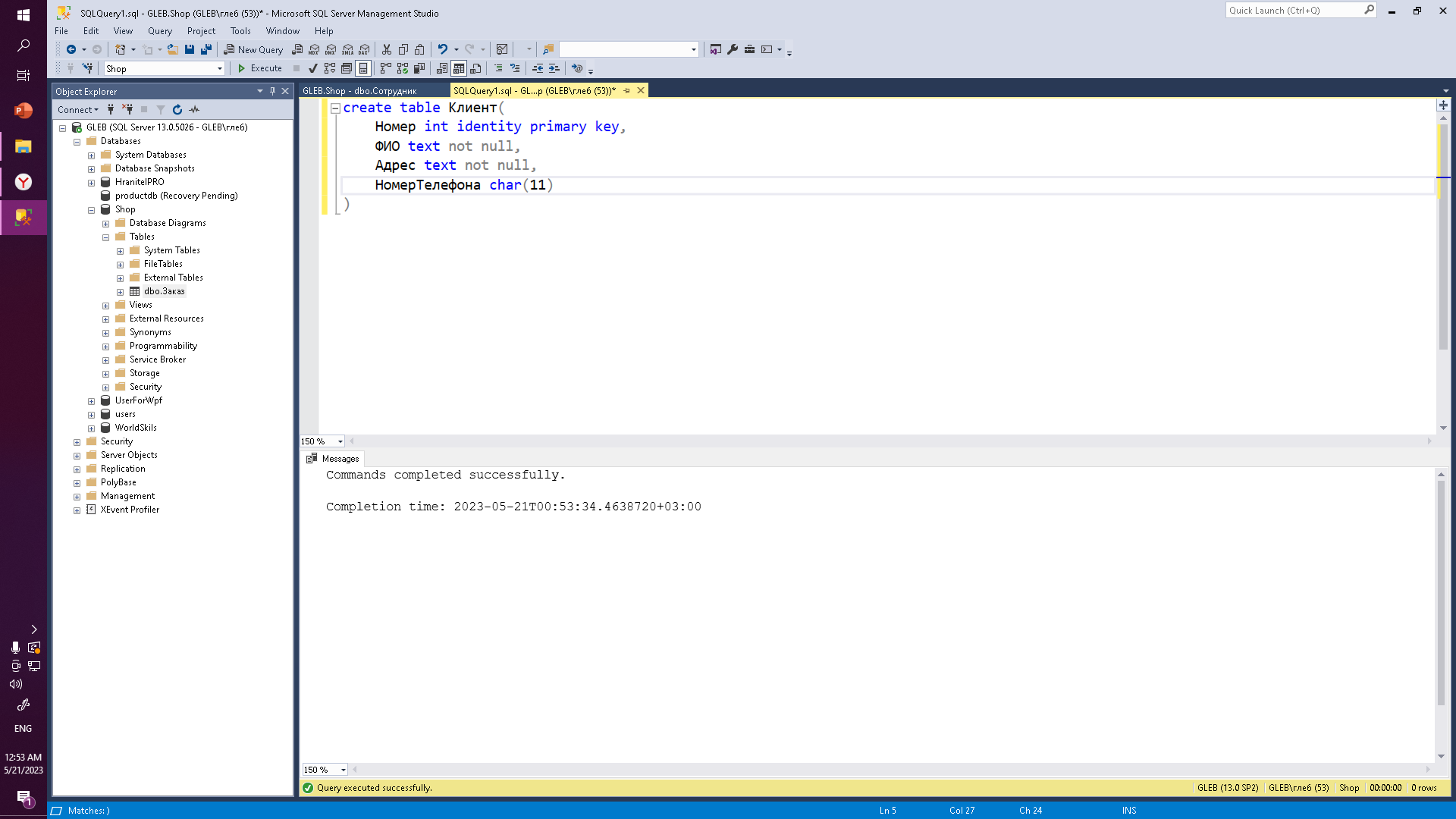
Создание таблицы Заказ

Следующей таблицей, которую предстоит создать будет таблица Заказов. Она будет содержать 8 полей (Номер Заказа, Общая Стоимость, Дата Заказа, Время Заказа, Дата И Время Поставки, Сотрудник, Клиент, Статус)

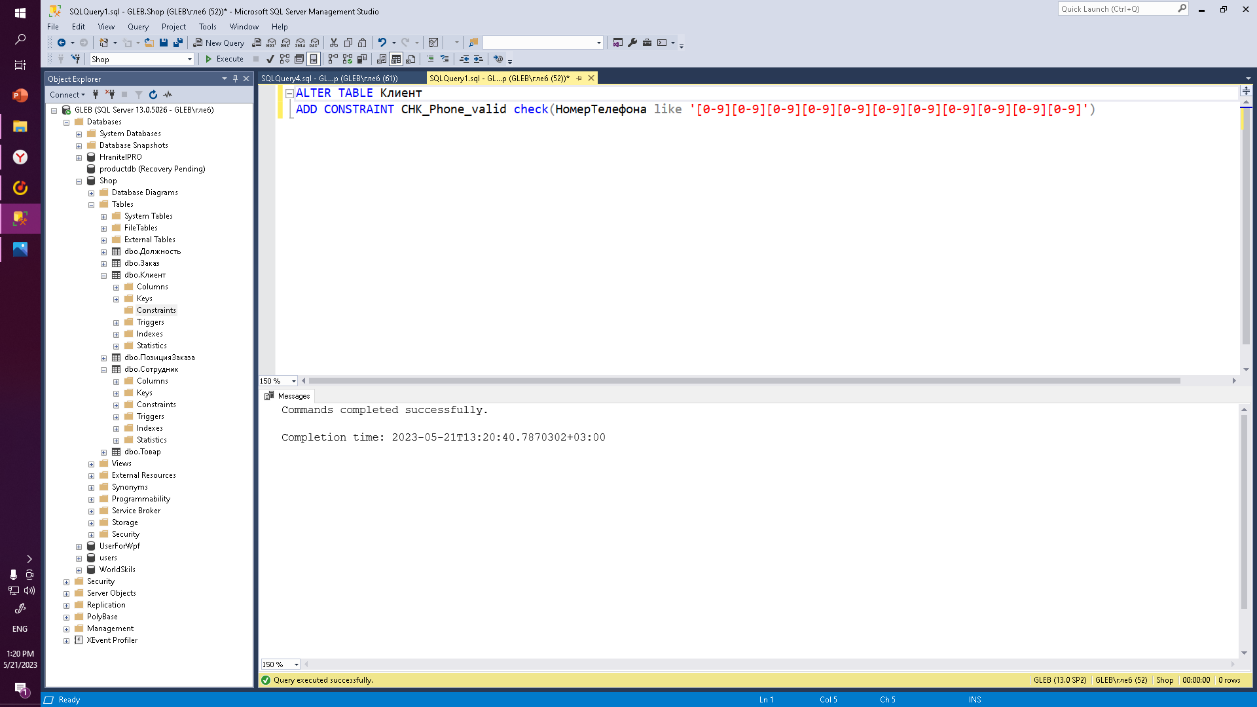
Поле номер заказа будет иметь целочисленный тип для того, чтобы соединить таблицу с последующими. Поле общая стоимость имеет денежный тип и необходимо для подсчета общей стоимости заказа (Цена из таблицы товар умноженное на количество из таблицы позиция заказа). Дата заказа имеет тип данных date, чтобы отображать дату заказа товара. Время заказа имеет тип данных time, чтобы отображать время заказа товара. Поле дата и время поставки имеет тип данных datetime (это тип который одновременно содержит дату и время) и оно нужно для записи данных о поставке. Поле сотрудник имеет целочисленный тип, чтобы потом эту таблицу соединить эту таблицу с таблицей сотрудников. Поле клиент имеет тот же тип для того, чтобы соединить текущую таблицу с таблицей клиентов. Поле статус такого же типа для соединения с одноименной таблицей

Создание таблицы Клиент

Перейдем к создание таблицы клиентов, в которой будет отображаться информация о клиентах и она будет содержать 4 поля(Номер, ФИО, Адрес, Номер телефона)

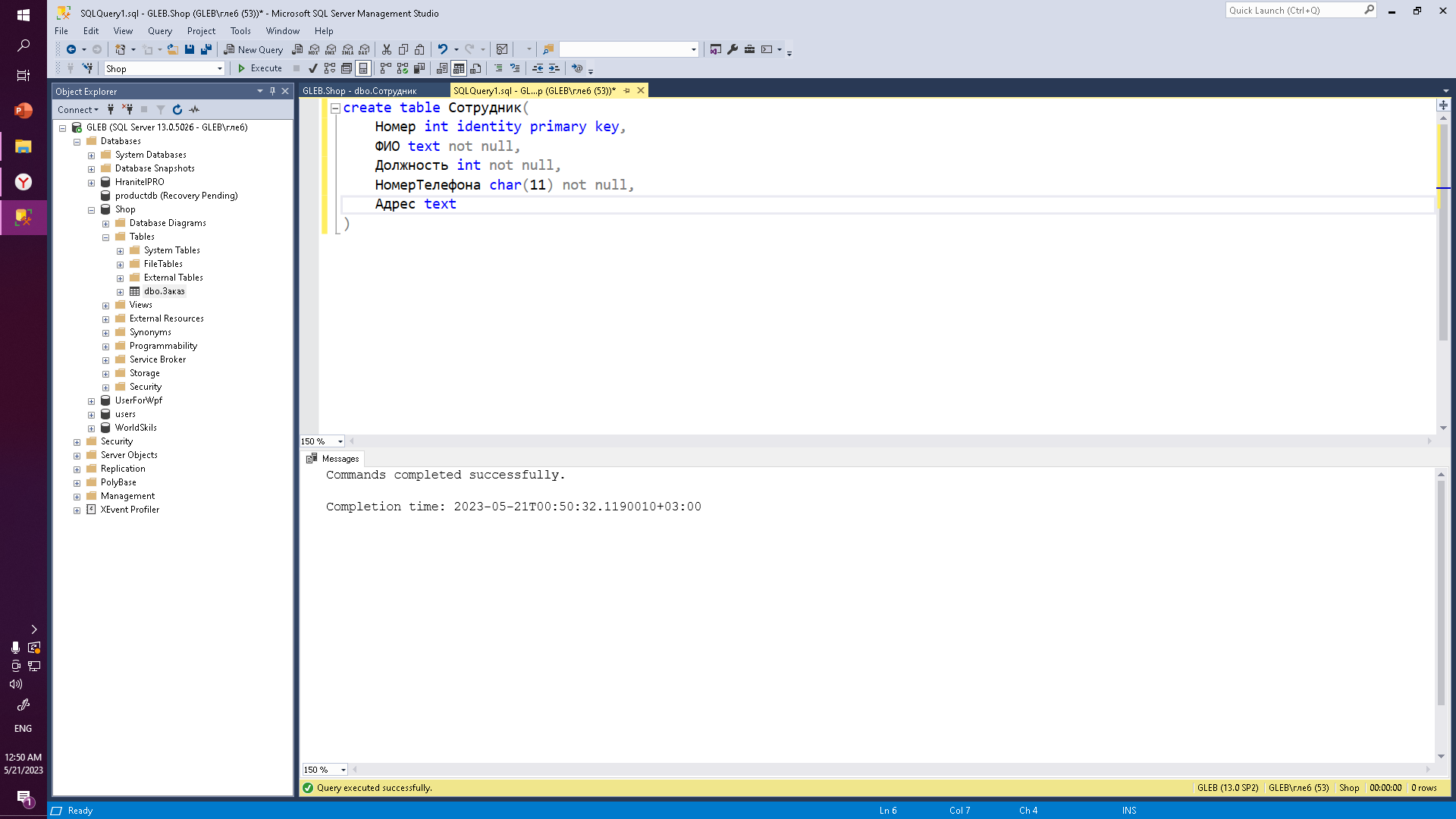


Поле номер с типом целочисленным данных будет соединять эту таблицу с таблицей заказов. Поле ФИО с текстовым типом данных будет содержать имя, фамилию и отчество клиента. Поле адрес с текстовым типом данных будет содержать информацию об адресе клиента. Поле номер имеет символьный тип данных и будет содержать номер телефона клиента.

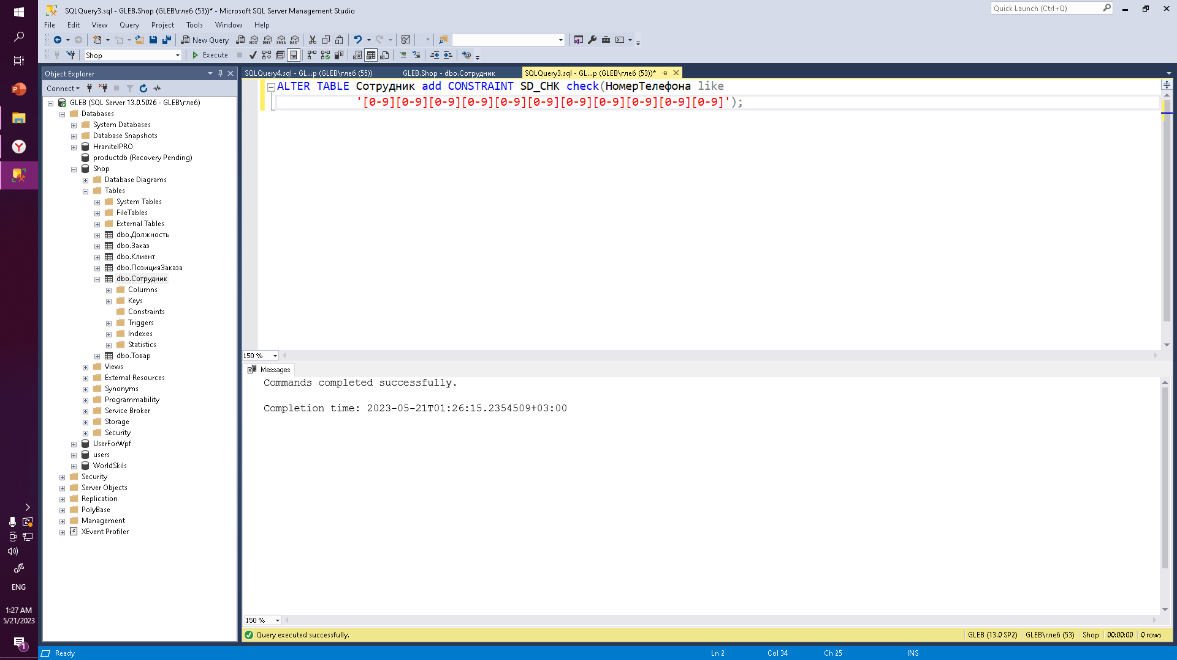
Для корректного ввода номера телефона необходимо добавить маску

Создание таблицы Сотрудник

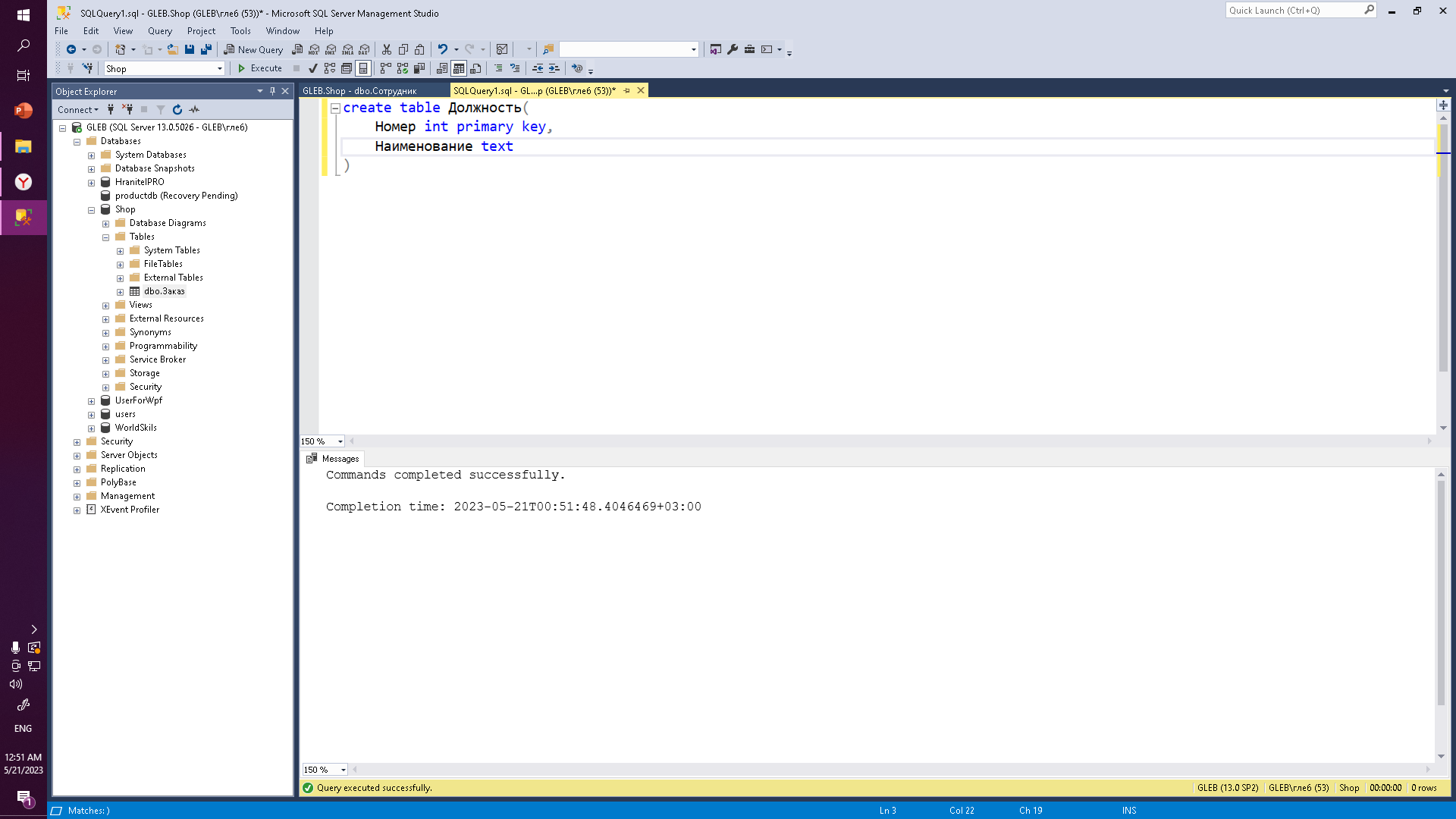
Перейдем к созданию таблицы сотрудник, содержащую необходимую информацию о сотрудниках, она содержит 5 полей(Номер, ФИО, Должность, Номер Телефона, Адрес)



Поле номер с целочисленным типом данных соединяет данную таблицу с таблицей заказов. Поле ФИО имеет текстовый тип данных и содержит имя, фамилию и отчество сотрудника. Поле должность имеет целочисленный тип данных для соединения с одноименной таблицей. Поле номер телефона с символьным типом данных будет хранить номер телефона сотрудника. Текстовое поле адрес будет хранить адрес сотрудника

Для корректного ввода номера необходимо добавить маску

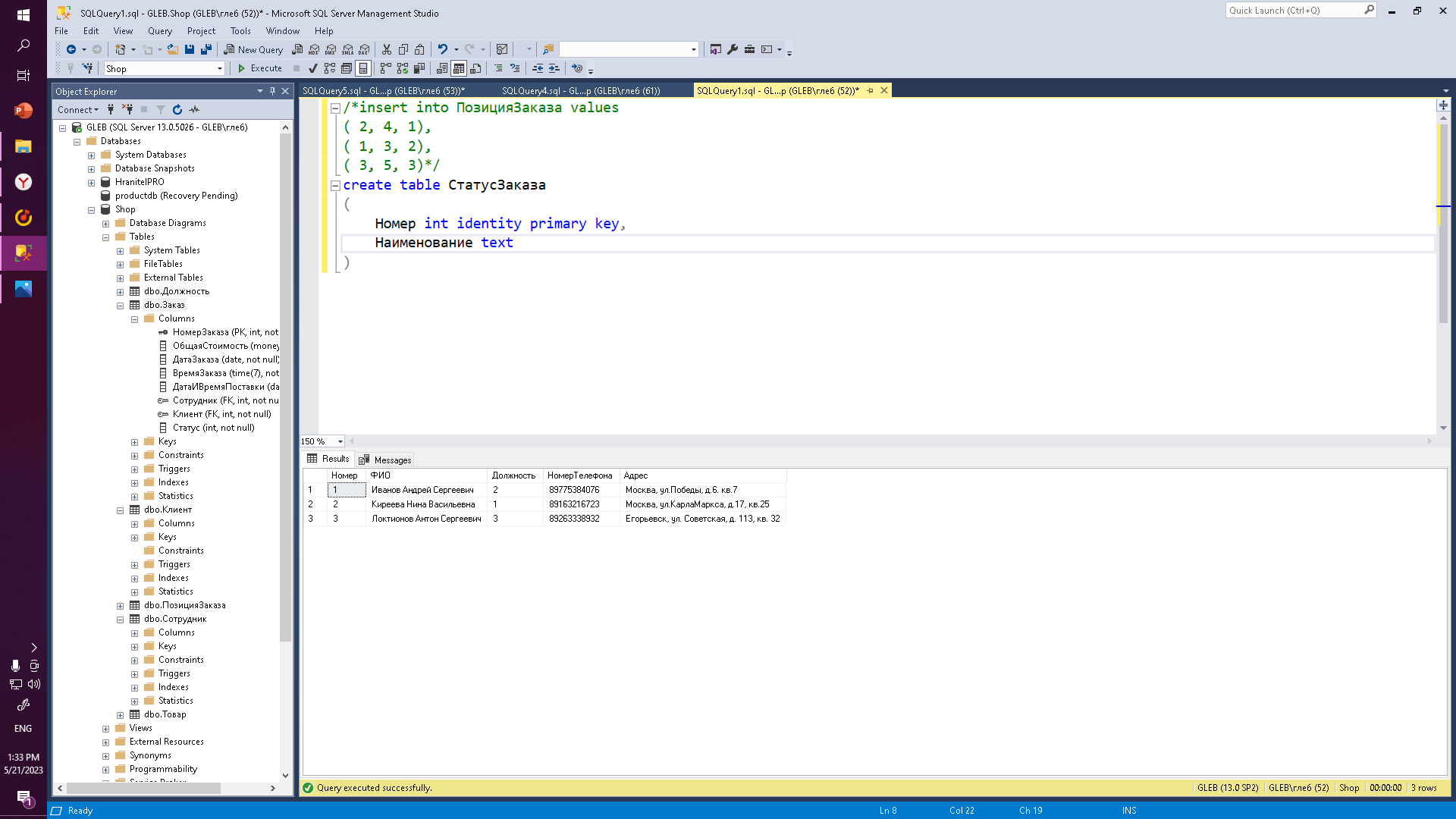
Создание таблицы Должность

Следующей создадим таблицу должностей, в которой будет всего 2 поля (Номер, Наименование) и в ней будут данные о должности сотрудников

Поле номер с целочисленным типом данных необходимо для соединения данной таблицы с таблицей сотрудников. Текстовое поле наименование будет содержать в себе название должности

Создание таблицы Статус Заказа

Заключительной таблицей создадим таблицу, содержащую статусы заказов, она будет содержать всего 2 поля(Номер, Наименование)



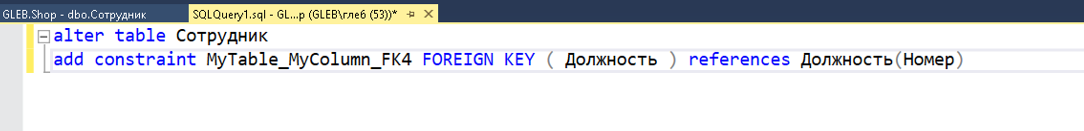
Поле номер имеет целочисленный тип для соединения текущей таблицы с таблицей сотрудников. Текстовое поле наименование содержит информацию о статусе заказа

Создание связей между таблицами

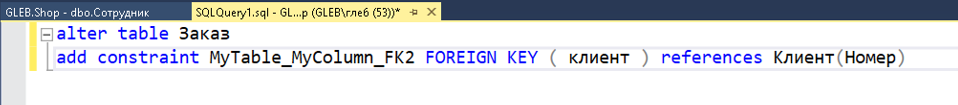
Чтобы создать связи между таблицами воспользуемся внешними ключами, для которых уже были подготовлены соответсвенные поля в таблицах. Для создания внешних ключей будет использоваться команда alter table:



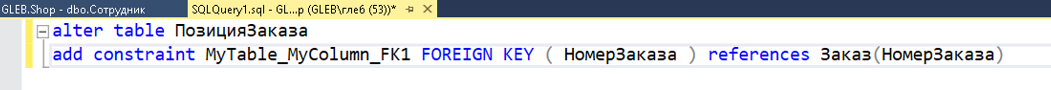
* Связь между Должностями и Сотрудниками



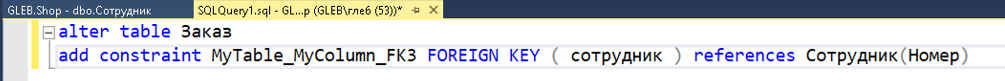
* Связь между Клиентами и Заказами



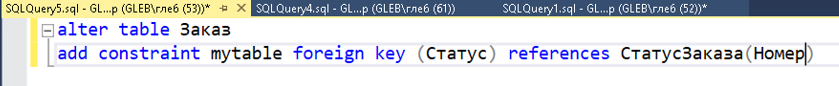
* Связь между Заказами и Позицией заказов



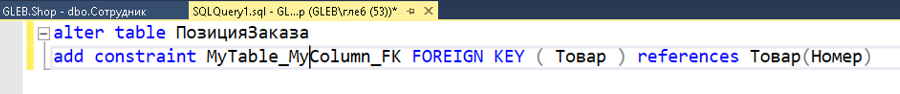
* Связь между Заказами и Сотрудниками



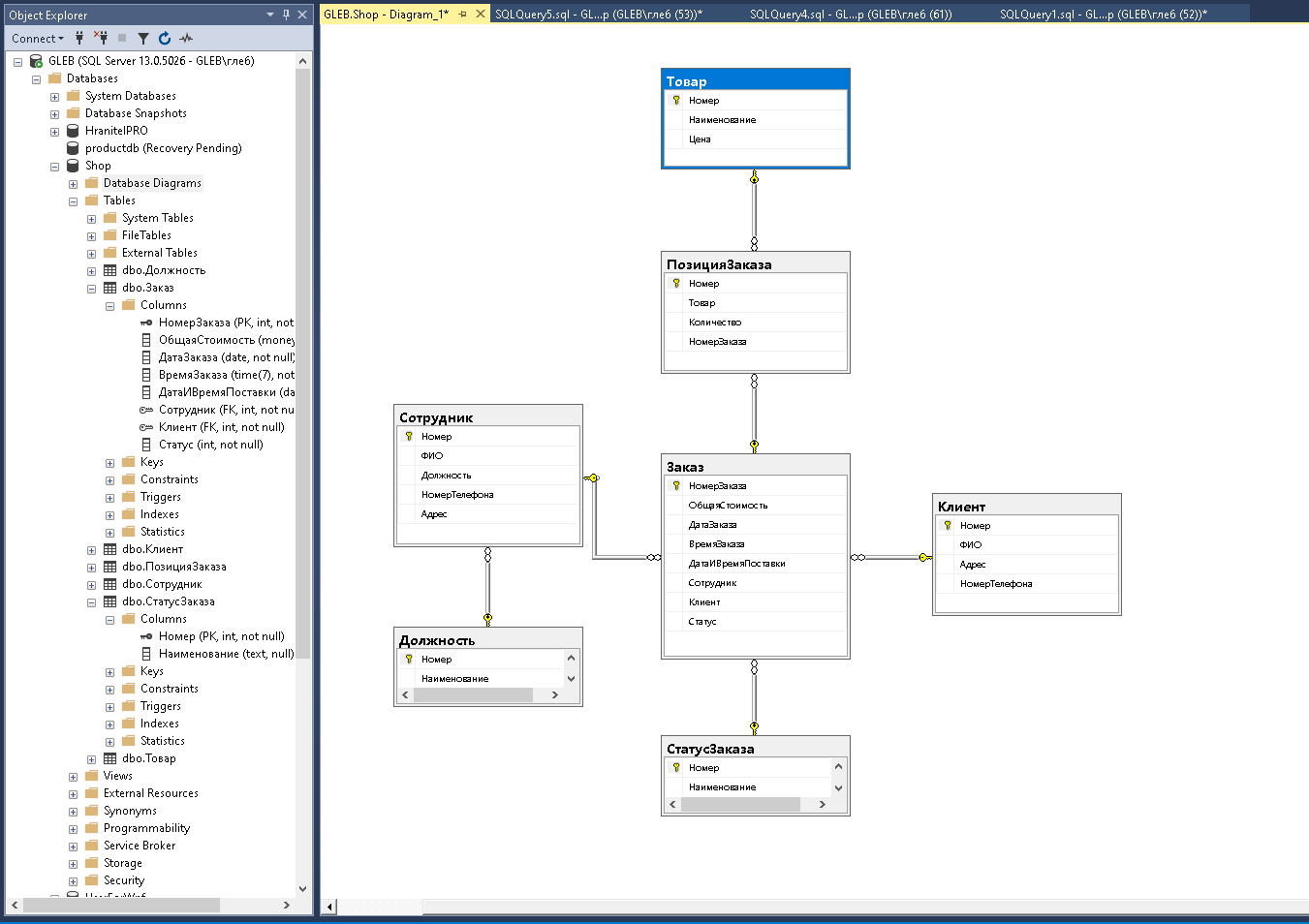
* Связь между Заказами и Статусами заказов



* Связь между Позицией заказов и Товарами



По итогу создания таблиц и связей между ними мы получаем такую базу данных:

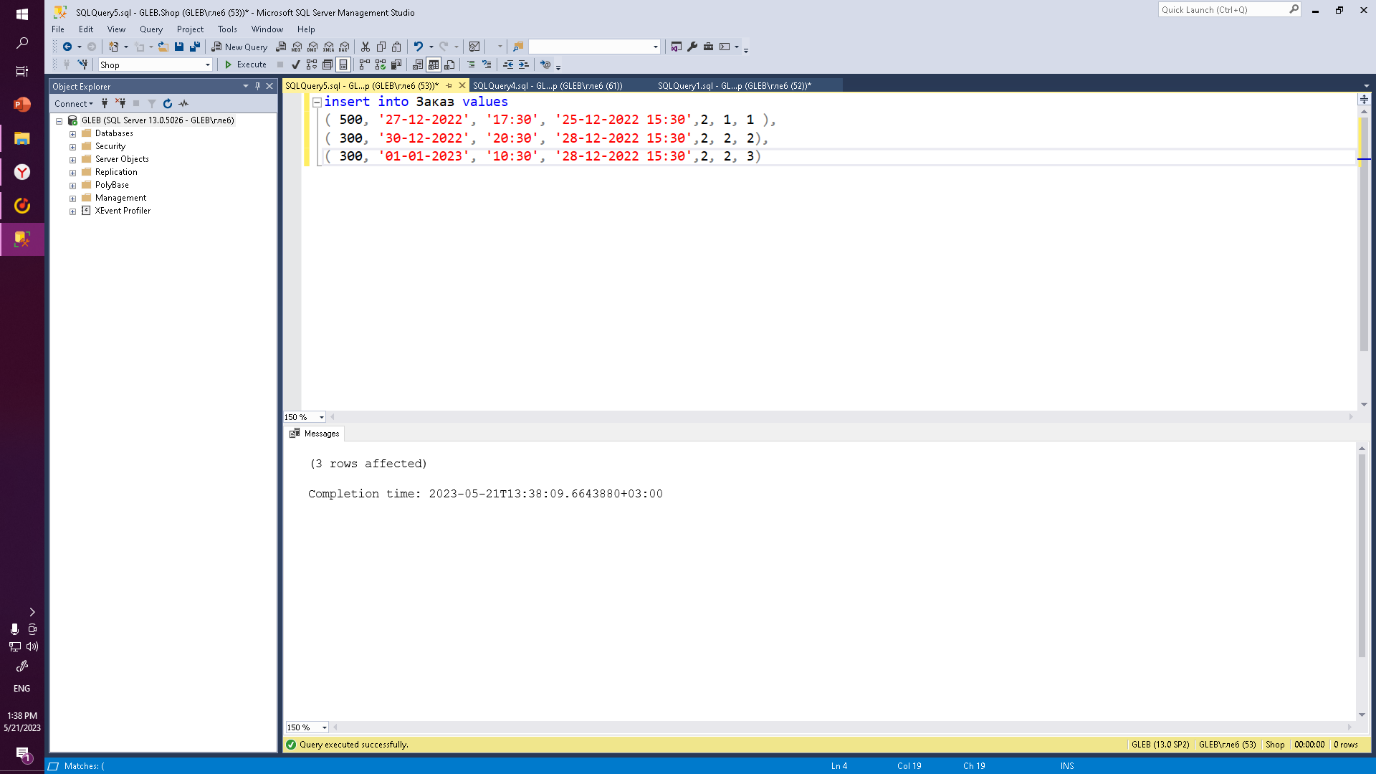


Добавление данных в таблицы

Для добавления данных в таблицы воспользуемся командой insert into, синтаксис которой выглядит так:

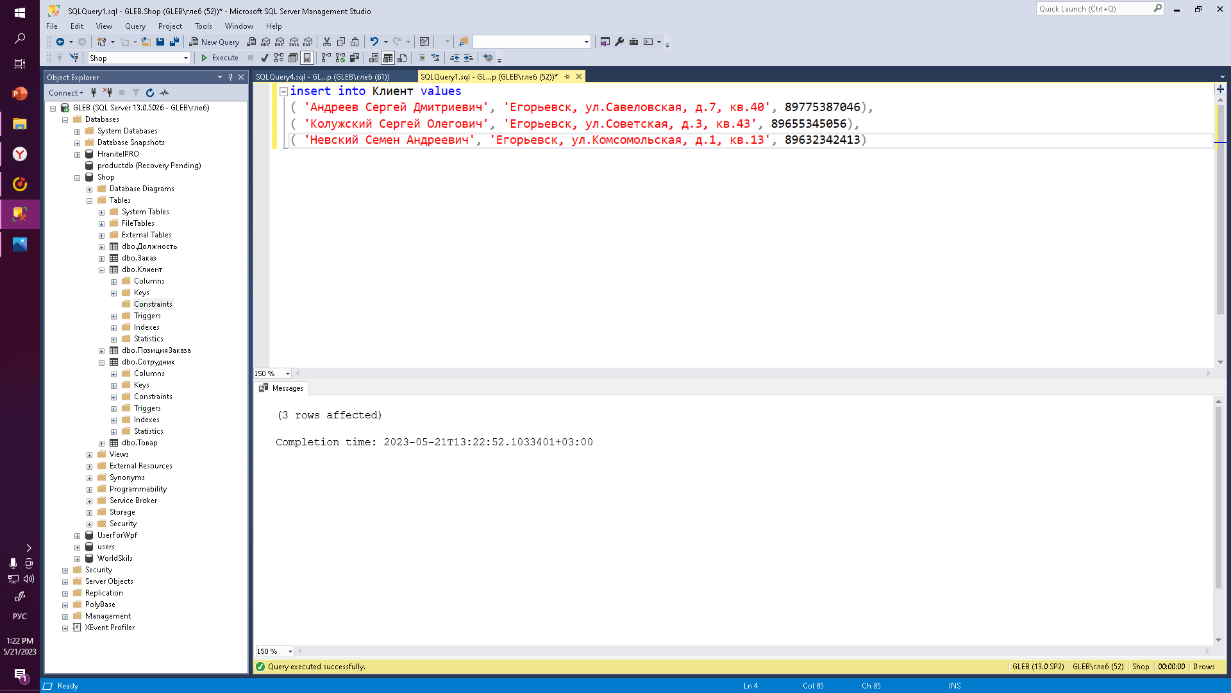


Заполнение таблицы заказов



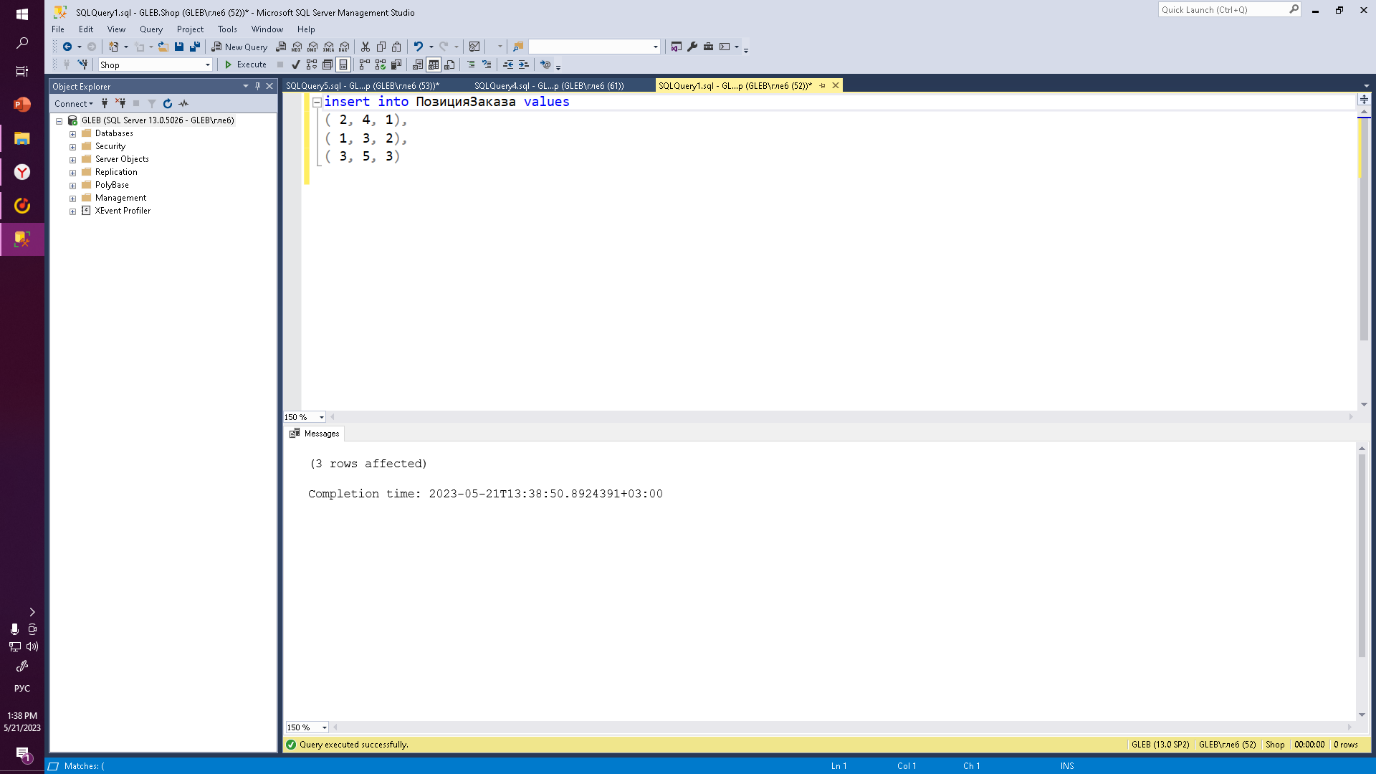
В этой записи первым идёт общая стоимость заказа, а не его номер, так как поле номер имеет свойство identity – то есть при формировании каждой новой записи, номер заказа формируется сам, дата заказа, время заказа, дата и время поставки, номер сотрудника, номер клиента, номер статуса заказа.

Заполнение таблицы клиентов



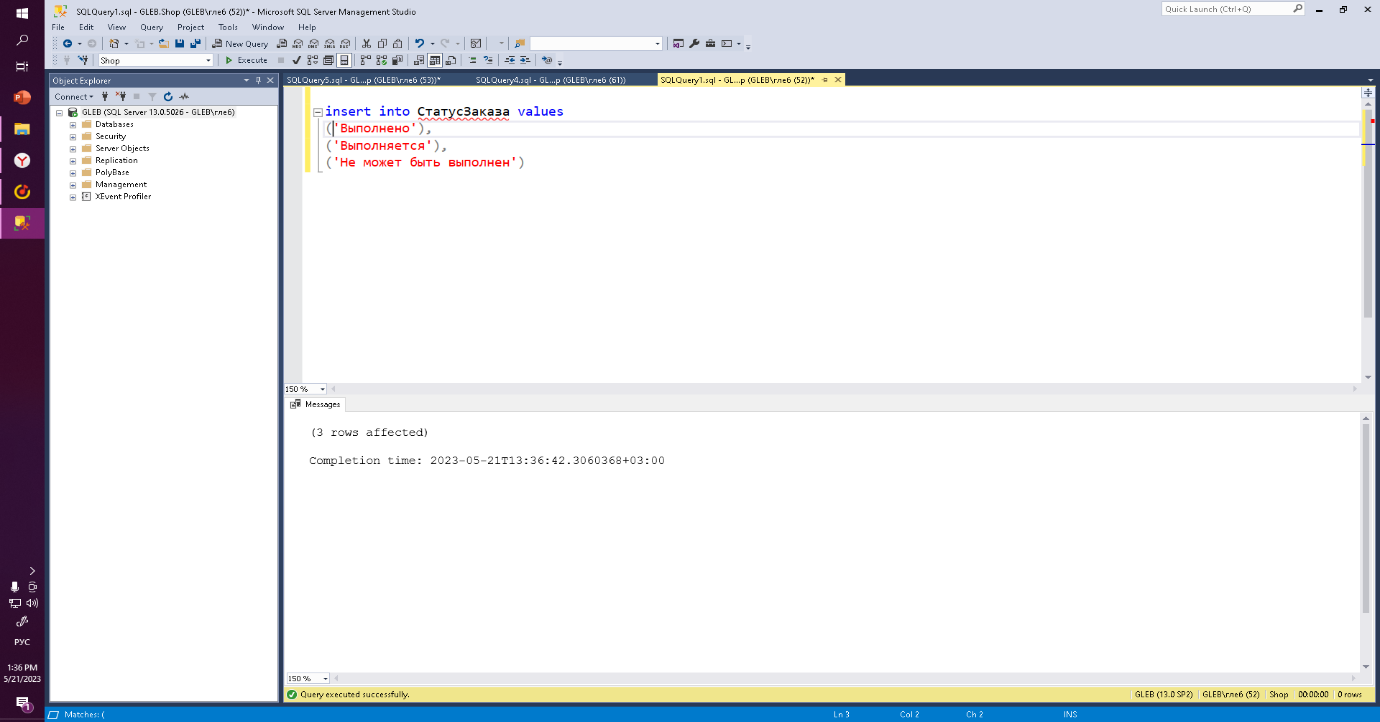
В текущей записи первым аргументом является ФИО клиента, а не номер по аналогичной причине (Поле номер имеет свойство identity), потом идет адрес проживания клиента, а затем его номер телефона

Заполнение таблицы позиций заказов



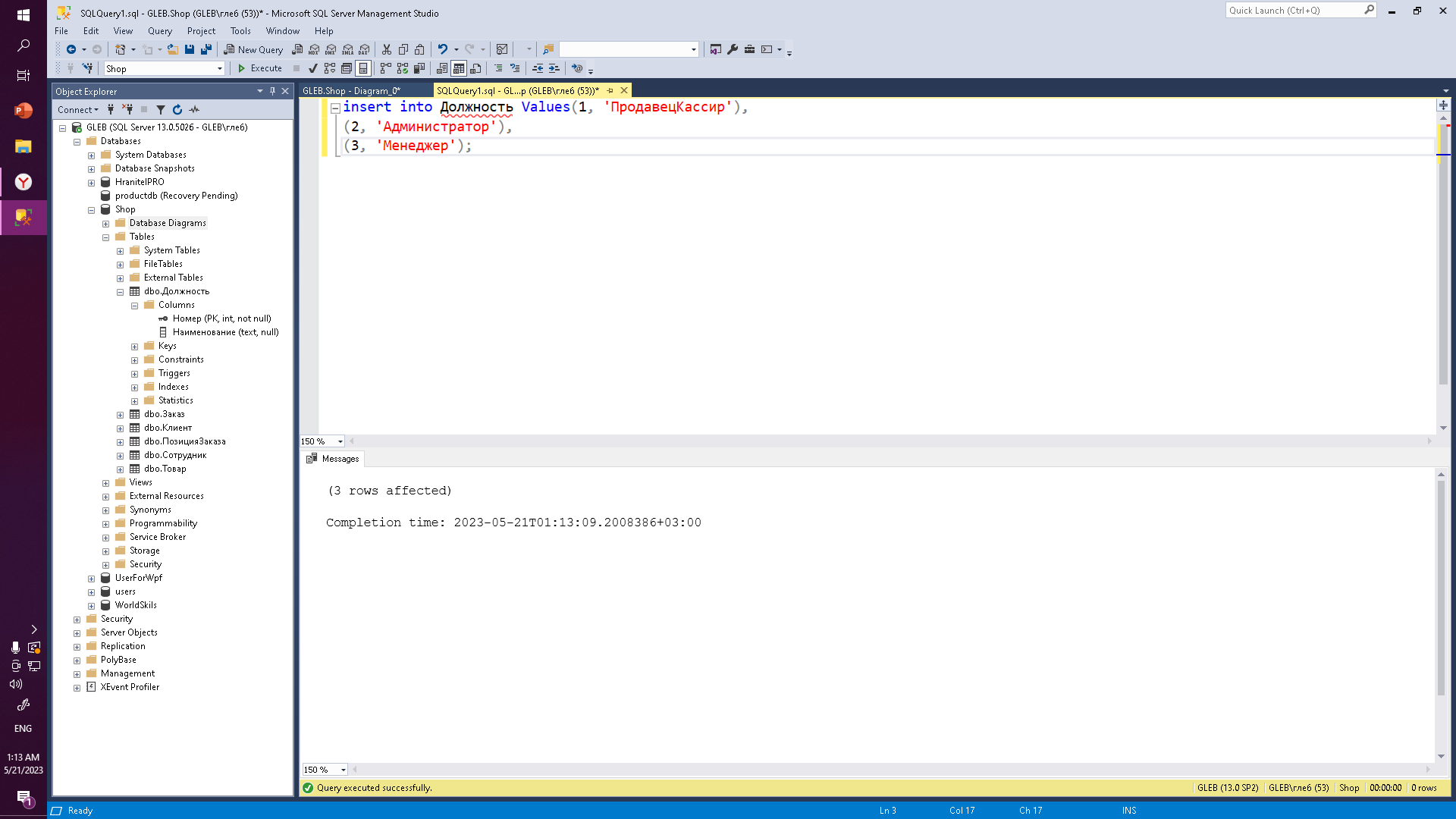
Номер снова не заполняем по той же причине и получается, что первым аргументом идет номер товара, вторым- его количество, а третьим- номер заказа

Заполнение таблицы статусов заказов



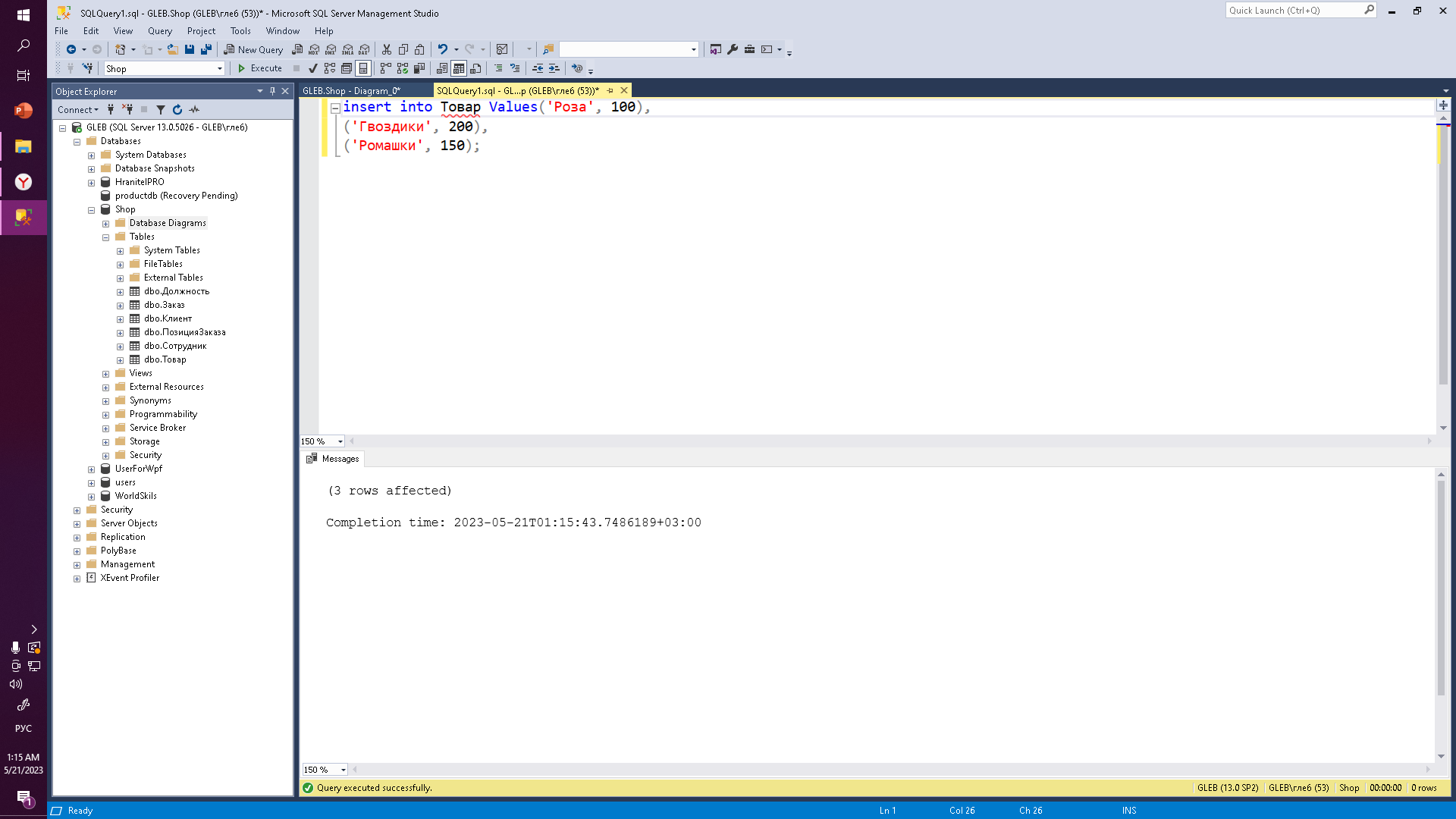
Снова не вводим номер. Заполняем именования статусов заказа

Заполнение таблицы должностей



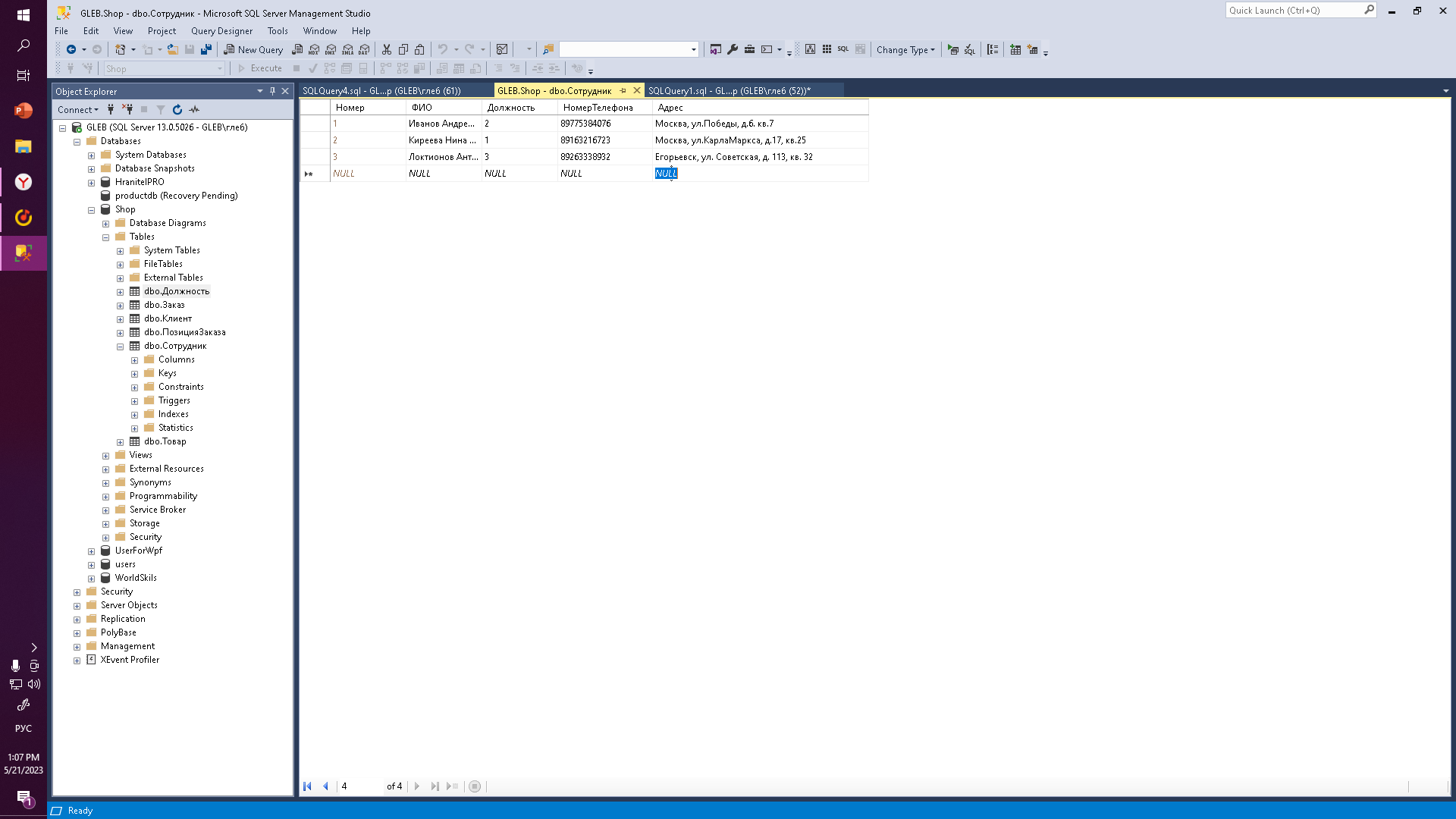
В текущей таблице первым аргументом идет номер должности, а вторым её именование

Заполнение таблицы товаров

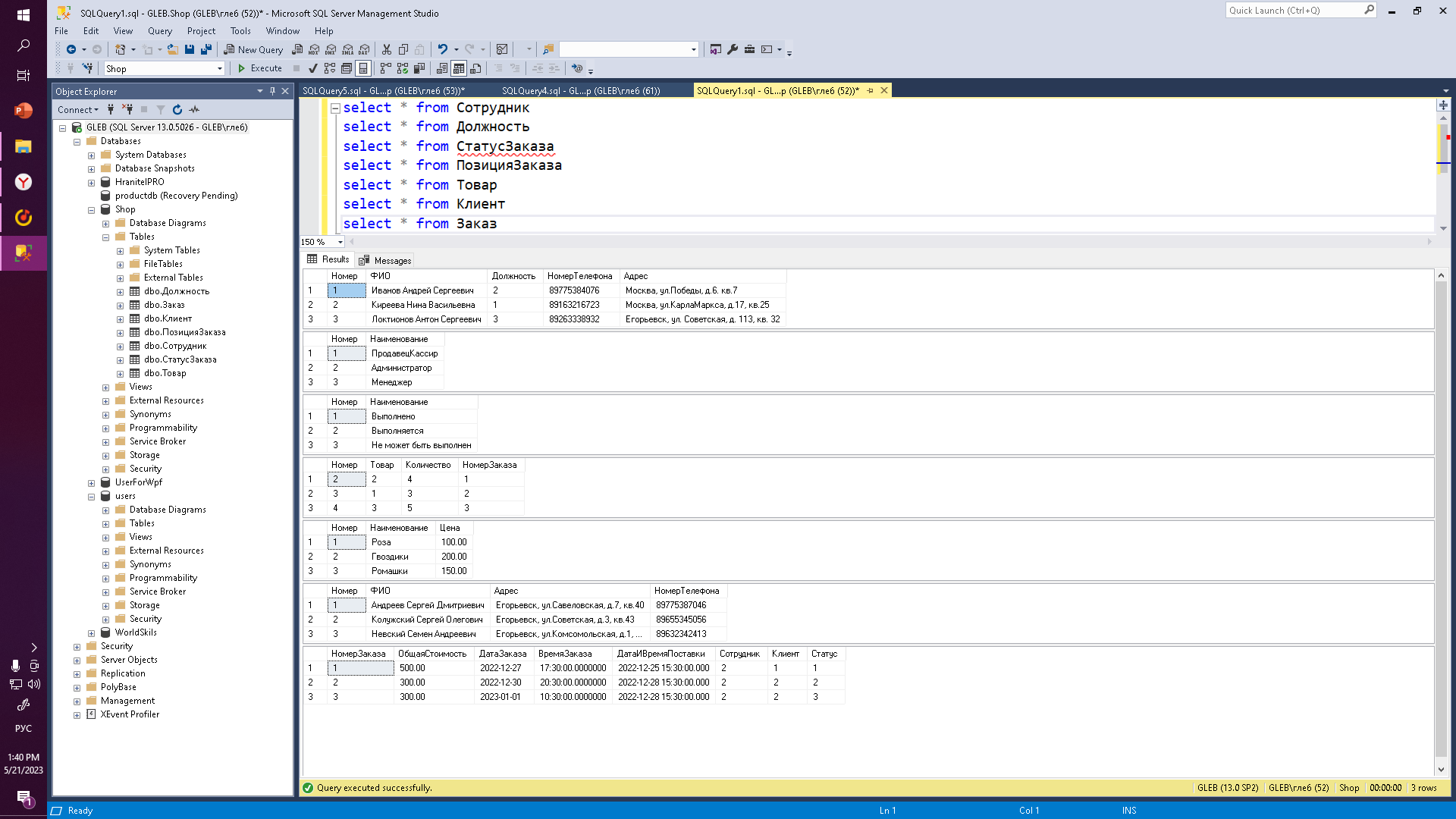


В данной таблице не заполняем номер тогда первым аргументом будет название товара, вторым- его стоимость

Заполнение таблицы сотрудников



Для заполнения данной таблицы был использован визуальный интерфейс MSsql. Для заполнения таблицы данным способом необходимо в обозревателе объектов раскрыть базы данных, найти и раскрыть нужную базу, найти и раскрыть таблицы, нажать ПКМ и выбрать изменить первые 200 строк и тогда в обозревателе решений появится данное окно, где можно будет внести все необходимые данные без использования запросов

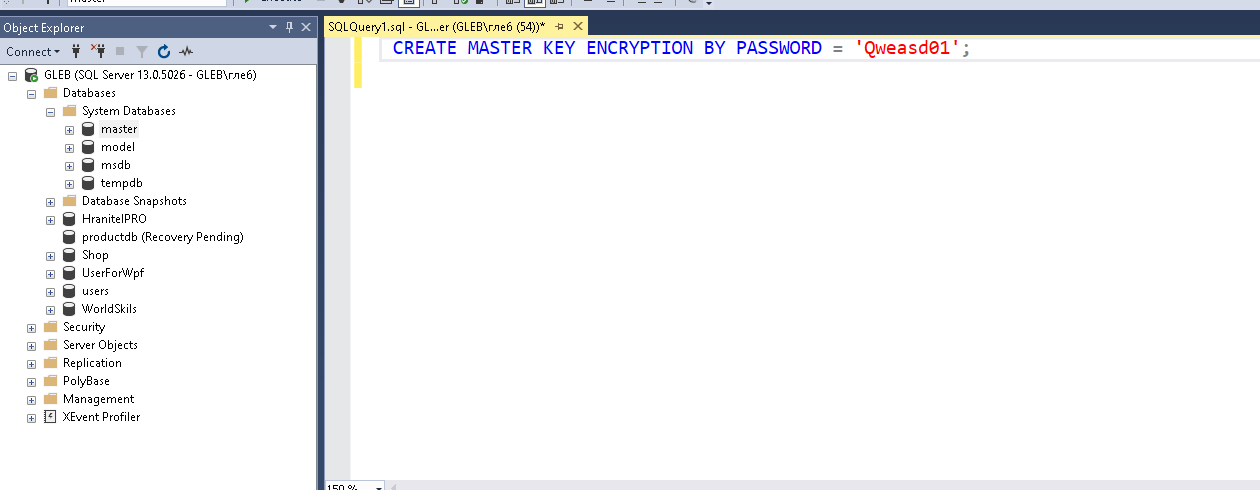
После внесения всех данных с помощью запроса select \* from «название таблицы», выведем все таблицы, чтобы убедиться, что все таблицы заполнены

Создание шифрования при помощи средств MsSql

Для шифрования резервной копии необходимо указать алгоритм шифрования и шифратор для защиты ключа шифрования. Поддерживаются следующие варианты шифрования:

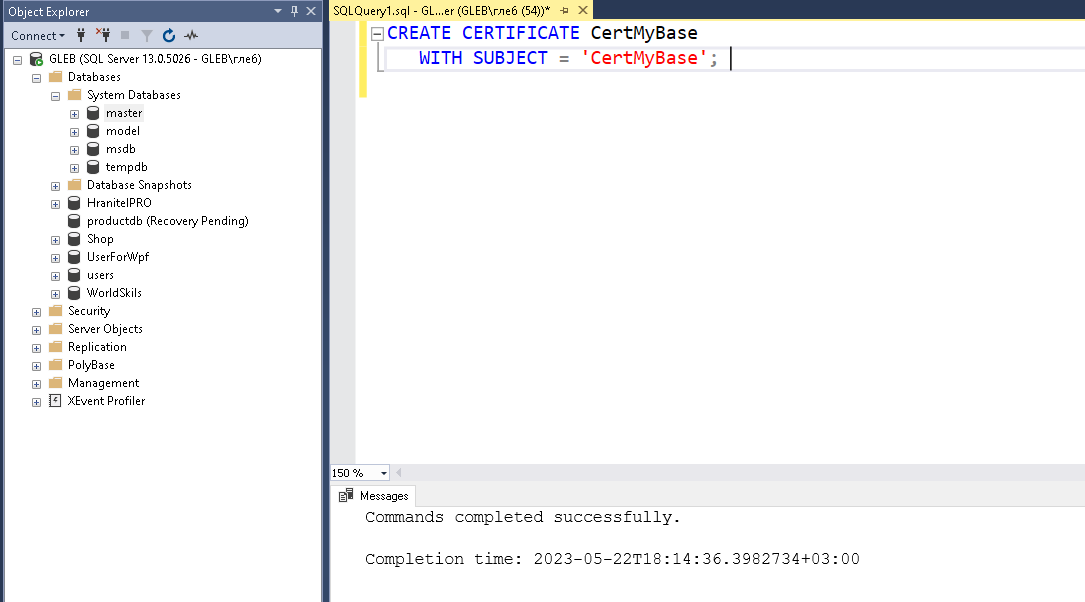
* **Алгоритм шифрования:** поддерживаются следующие алгоритмы шифрования: AES 128, AES 192, AES 256 и Triple DES.
* **Шифратор:** сертификат или асимметричный ключ.

Создание мастер ключа



Первым этапом для шифрования данных идет создание мастер ключа. Его задача- защита закрытых ключей сертификатов и асимметричный ключей, которые есть в базе данных. Он необходим, так как шифрование данных будет производится на основе сертификата. Мастер ключ создается при помощи команды CREATE MASTER KEY, а его пароль при помощи ENCRYPTION BY PASSWORD = ‘Ваш пароль’, в этом случае Qweasd01

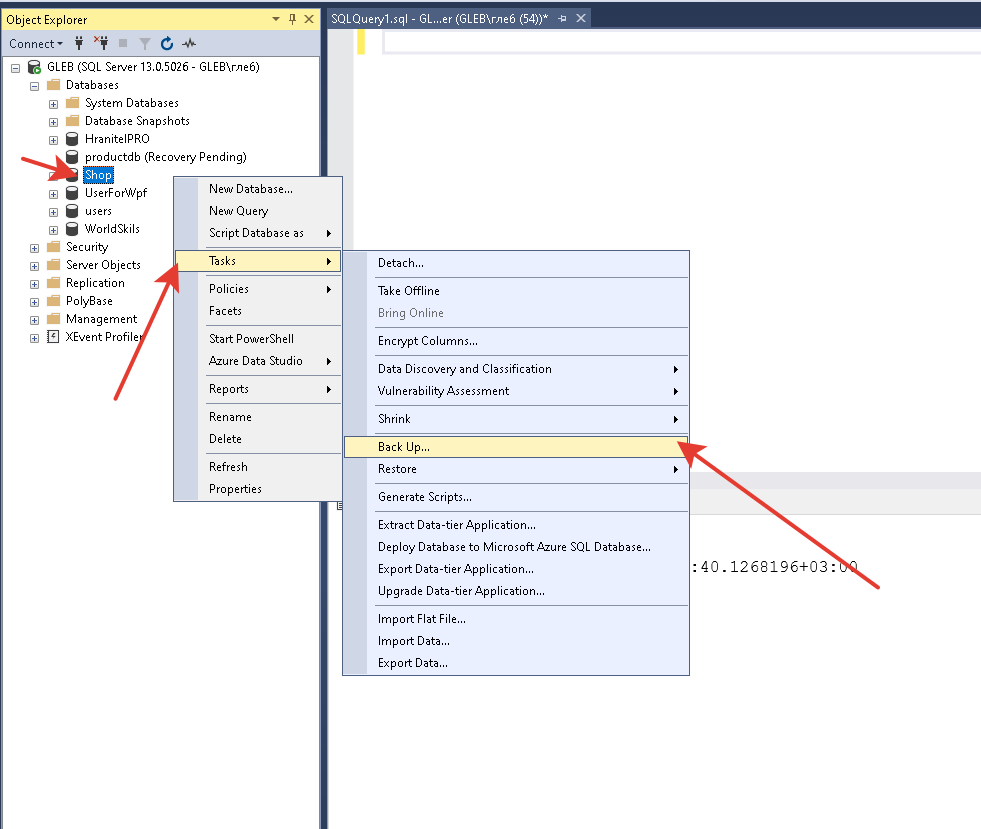
Создание сертификата



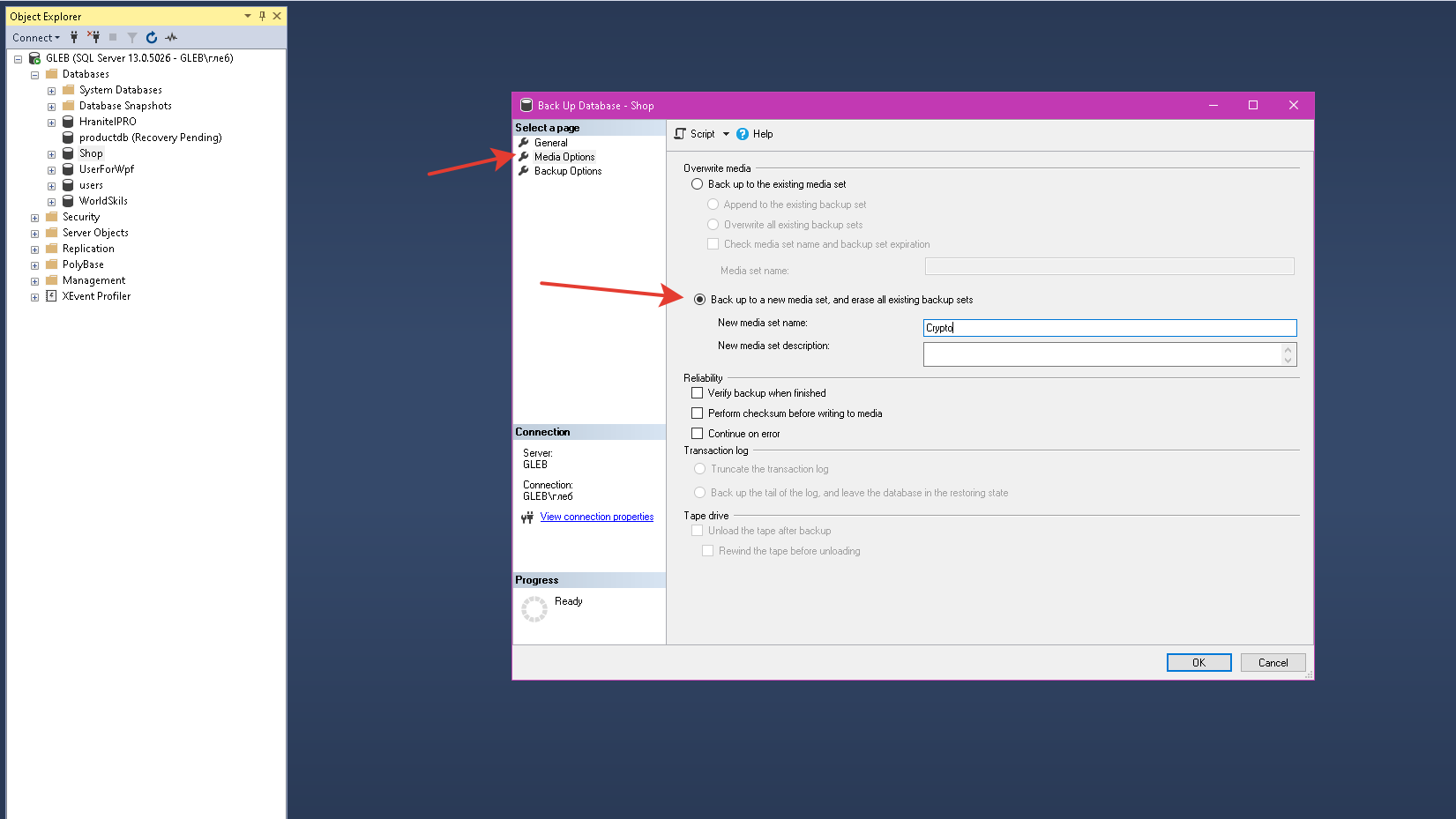
Вторым этапом шифрования базы данных идет создание сертификата, по которому в последствии будет происходить шифрование. Для его создания воспользуемся командой CREATE CERTIFICATE ИМЯ, в данном случае CertMyBase. Также создадим аргумент SUBJECT, он относится к полю в метаданных сертификата, определяемому стандартом X.509

Резервное копирование зашифрованной базы данных

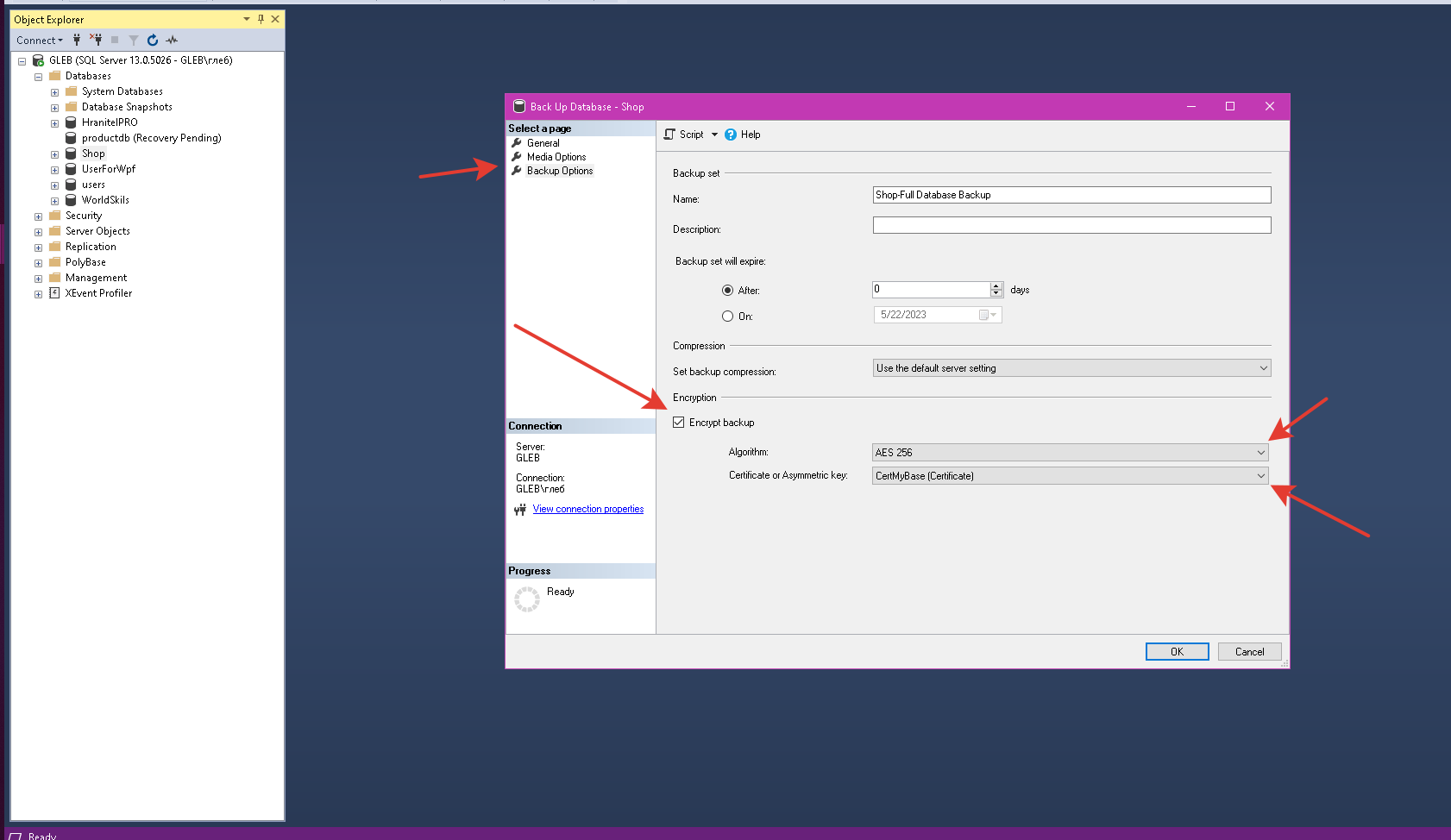
После всех подготовительных этапов можно переходить к резервному копированию зашифрованной базы данных.



Для этого мы правой кнопкой нажимаем на базу данных, которую нам нужно зашифровать. В выпадающем меню выбираем пункт задачи, в следующем выпадающем меню выбираем резервное копирование. После этих манипуляций откроется следующее окно:



В левом боковом меню выбираем настройки медиа, в открывшемся окне необходимо выбрать пункт «создайте резервную копию на новом носителе и удалите все существующие наборы резервных копий» и придумываем название нового носителя.



Переходим на вкладку ниже, то есть в параметры резервного копирования. В этом окне необходимо поставить галочку в пункте «шифрование резервной копии». В выпадающем меню алгоритмов выбираем AES256. После чего можем выбрать ранее созданный сертификат

Хеширование данных с помощью возможностей C#

Для демонстрации хеширования данных при помощи C# был подготовлен проект. Создание проекта заключает в себе две главных задачи:

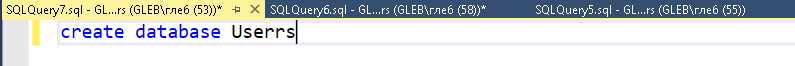
* Создание базы данных для хранения зашифрованной информации
* Создание приложения WPF

В свою очередь создание приложения WPF делится на подзадачи:

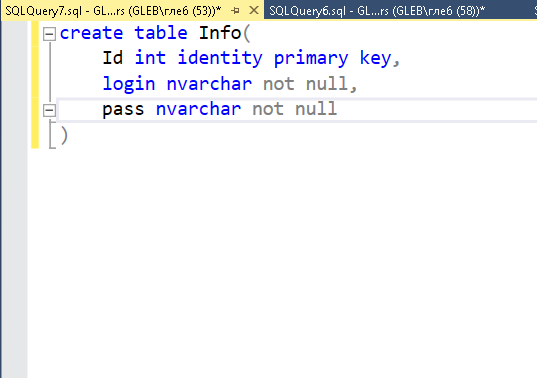
* Написание скрипта для хеширования данных
* Подключение базы данных
* Создание минимального пользовательского интерфейса

Создание базы данных

Для создания базы данных был использован MS Sql Server. Для создания таблиц использовалась команда Create database

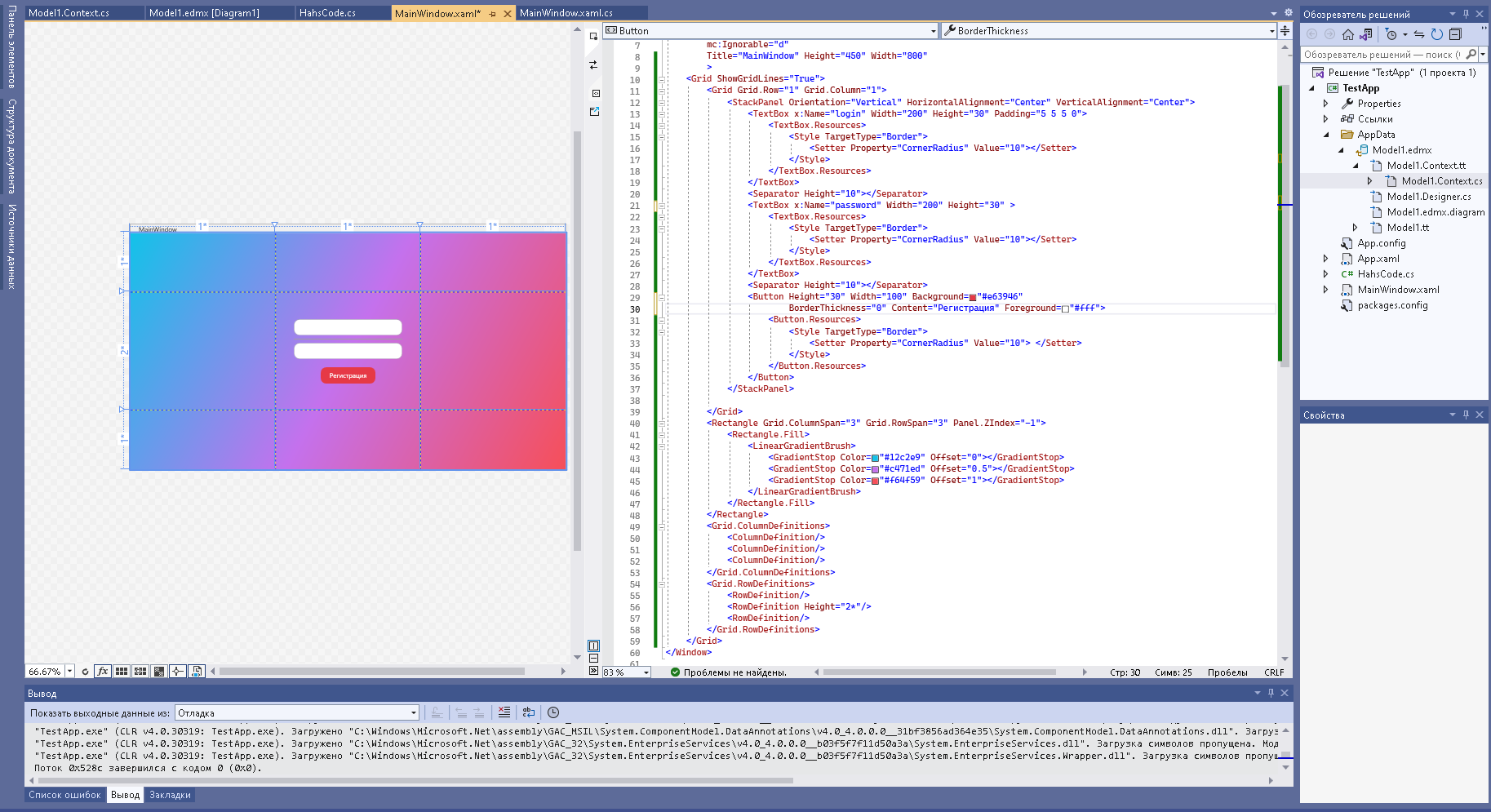


Теперь создадим таблицу с тремя полями. Поле Id будет хранить универсальный идентификатор пользователя, который генерируется сам при создании новой записи, благодаря аргументу IDENTITY. Поле логин, которое будет хранить логин пользователя. Поле пароль, которое будет хранить пароль в зашифрованном виде



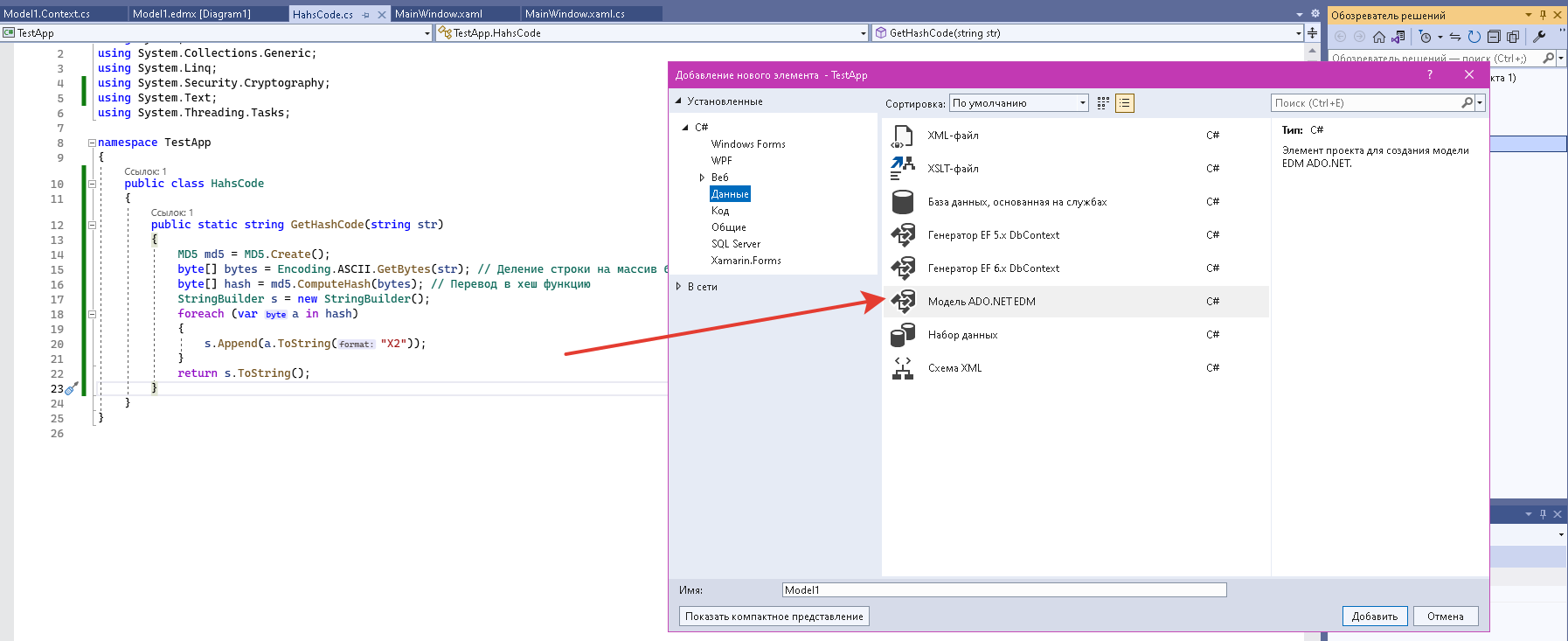
Создание приложения WPF

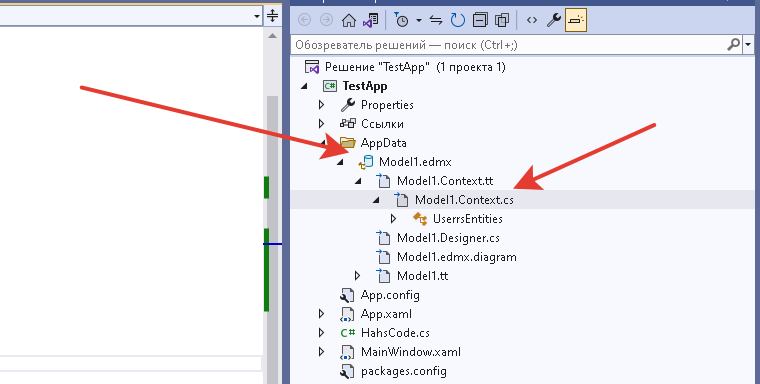
Первым делом открываем Visual Studio, кликаем создать проект, ищем проект с названием WPF и создаем его. После создания проекта создадим пользовательский интерфейс

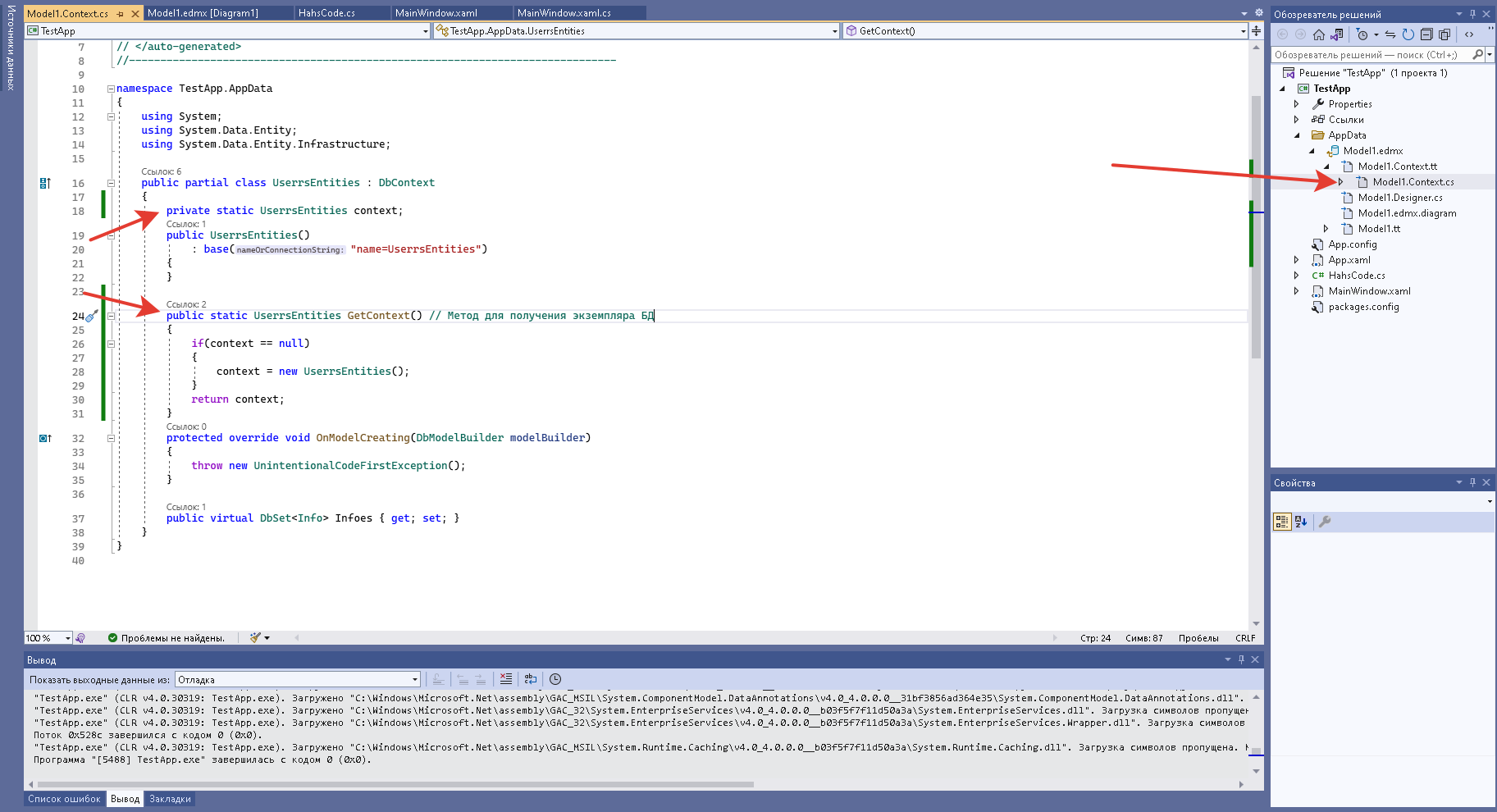


Интерфейс представляет из себя, один большой грид-контейнер, который делиться на три столбца и три строчки, в котором находиться еще один грид-контейнер, а в нем находится stackpanel в вертикальной ориентации. В stackpanel находятся два поля ввода, первый для ввода логина, второй для ввода пароля, а также есть кнопка, при нажатии на которую в базе данных будет появляться новый пользователь

После создания интерфейса перейдем к подключению базы данных. В Visual Studio для этого есть модель ADO.NET именно с ее помощью создается подключение к базе данных. Перед этим следует создать папку для хранения базы данных, в нашем случае она называется AppData

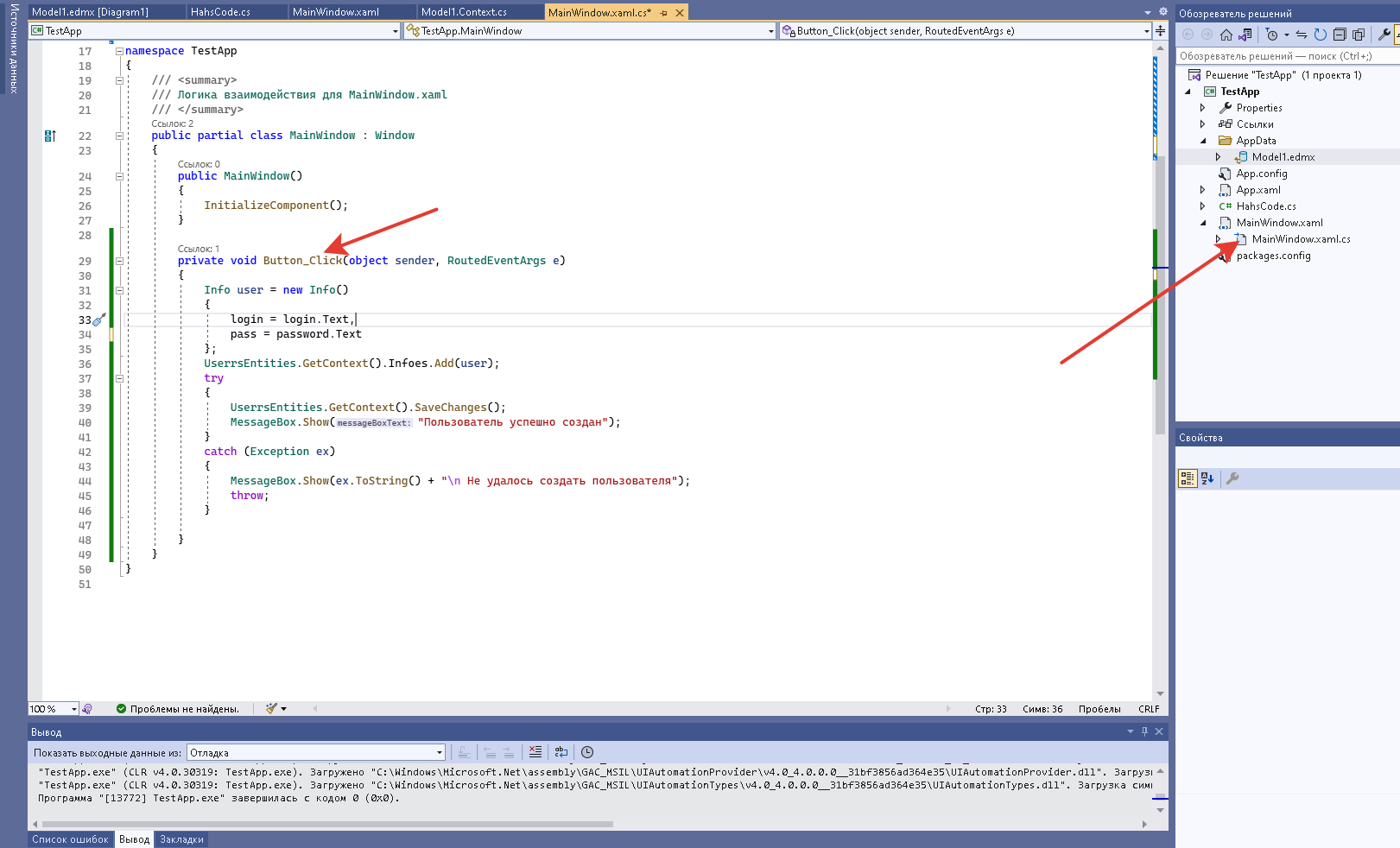


После проделанных действий Visual Studio сгенерирует классы для представления таблиц, а также контекст базы данных, который будет немного изменен для обращения к базе данных

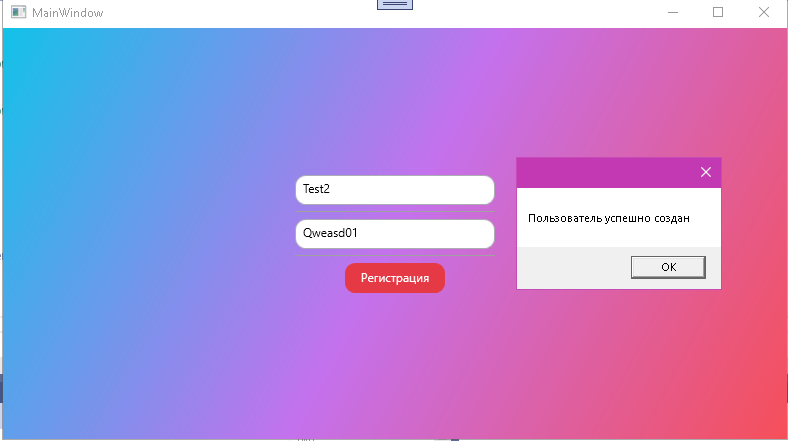
Следующем шагом надо модифицировать класс контекста базы данных:

Объявляем экземпляр базы данных с названием context. Для того, чтобы обращаться к базе данных создадим метод GetContext(), который будет проверять пустой ли наш экземпляр базы данных и если это так метод инициализирует экземпляр и вернет его нам

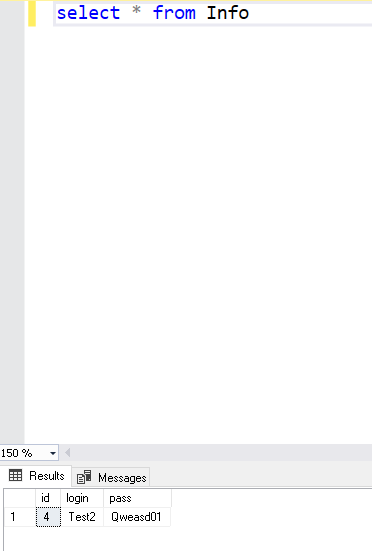
Теперь необходимо добавить логику добавление нового пользователя. Первым делом добавим обработчик нажатия кнопки, по нажатию которой будет включатся скрипт создания нового пользователя и добавления его в базу данных. Потом открываем файл MainWindows.xaml.cs, который является вспомогательным классом для интерфейса приложения, в нем ищем созданный обработчик событий



В обработчике событий создаем новый объект класса Info с именем user. Указываем, что значение логина будет равно значению текстового поля с именем логин, а пароль будет равен значению текстового поля с именем пароль. Потом обращаемся к базе данных вызываем метод ADD, который добавит нашего пользователя в базу данных. Добавление пользователя готово, осталось только сохранить изменения в базе данных для этого снова обращаемся к базе и вызываем метод SaveChanges(), чтобы приложение не упало при возникновении ошибки нужно переместить этот метод в блок TRY-CATCH. Теперь в случае успешного создания нового пользователя выскочит сообщение об успешном добавлении пользователя, в противном случае выскочит сообщение, что не удалось создать пользователя. Для тестирования функционала запустим приложение и попробуем добавить нового пользователя

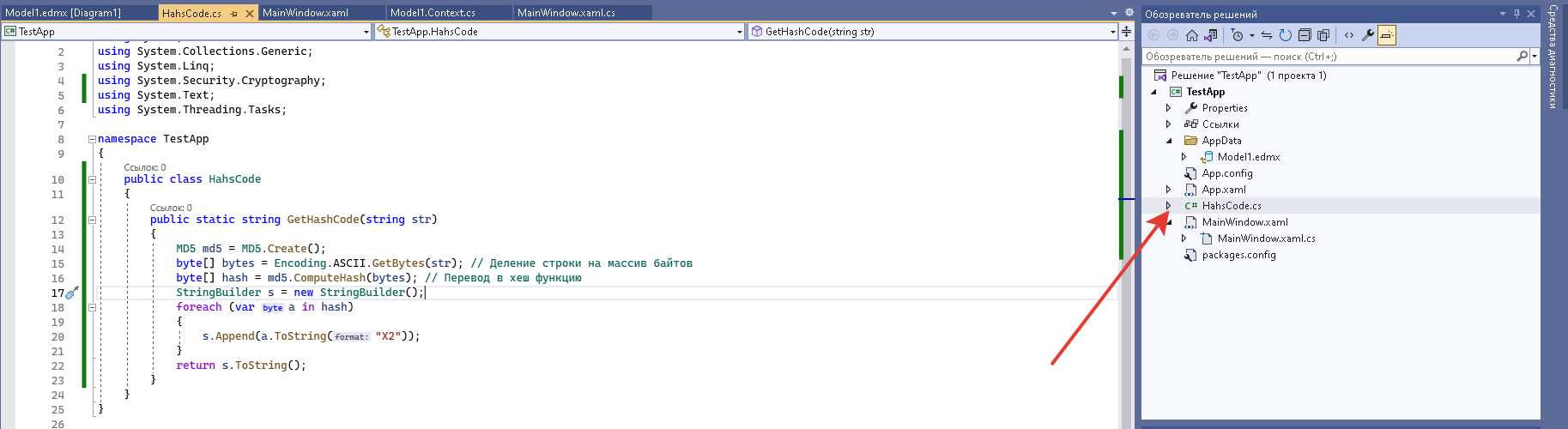


Проверим базу данных



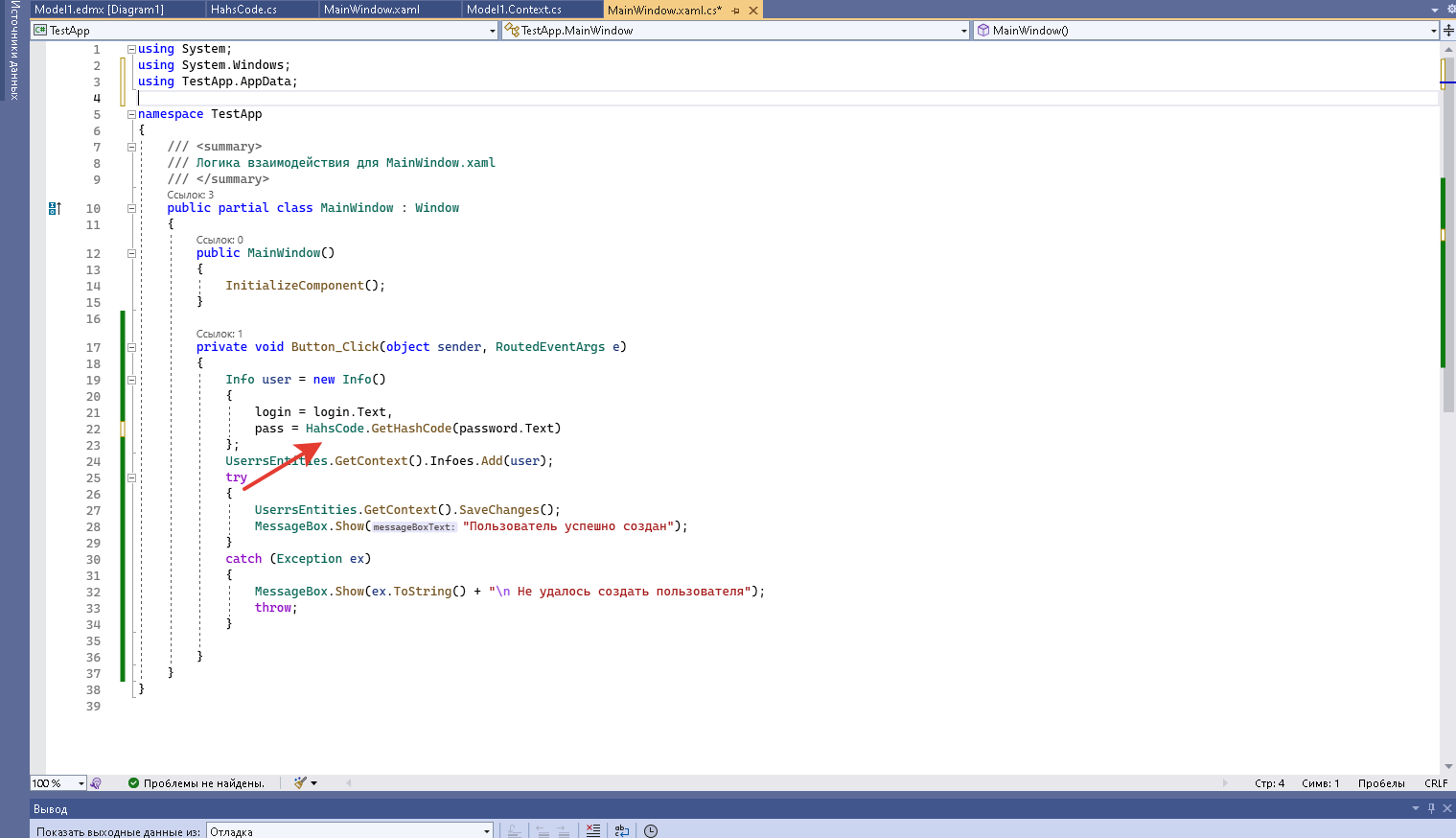
Данные успешно добавлены.

Последним шагом создания приложения будет добавление скрипта для хеширования пароля пользователя

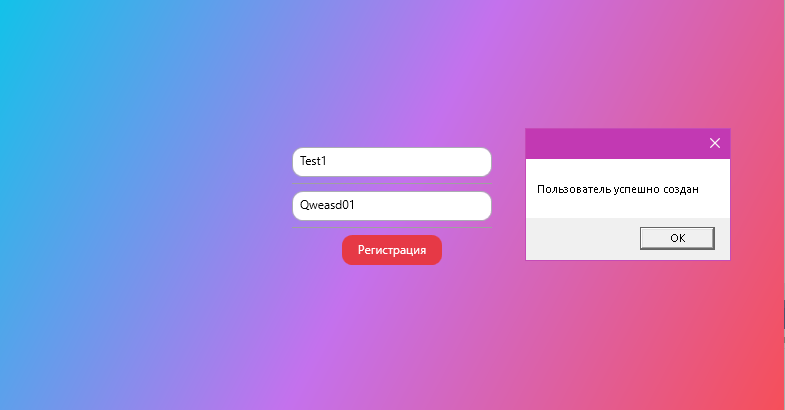


Для этого создаем новый класс с именем HashCode. В нем создаем метод GetHashCode(), который принимает строку и возвращает новую строку в виде хеш-кода. В теле метода создается объект класса MD5. Все алгоритмы хеширования работают с битами, поэтому входящую строку конвертируем в массив байтов. Потом создаем новый массив байтов, который будет хранить хеш-код и инициализируем его с помощью метода ComputeHash в который аргументом передаем строку, конвертированную в байтовый массив. Хеш-код готов, сейчас надо из массива байтов сделать строку, для этого используется объект класса StringBuilder, в который будет производиться запись итогового результата. Перебираем байтовый массив и с помощью метода append, записываем значение массива в созданный ранее объект класса StringBuilder и возвращаем его из метода

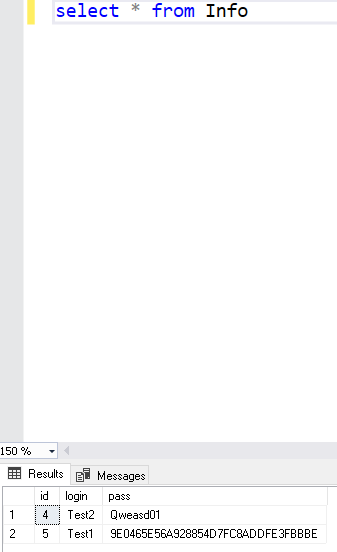
Класс готов. Для его использования нужно добавить созданный метод туда, где происходит запись пароля, то есть в MainWindow.xaml.cs



Протестируем финальную версию приложения!



Как видим вроде все также и ничего нового не произошло, но чтобы увидеть изменения надо посмотреть записи в базе данных



Были созданы два пользователя с разными логинами и одинаковыми паролями, но у второго пользователя вместо строки Qweasd01 идет хеш-код, это значит, что все успешно работает