# РАЗРАБОТКА КЛИЕНТСКИХ ПРИЛОЖЕНИЙ БАЗ ДАННЫХ

*Лабораторная работа 1*

# СОЗДАНИЕ БАЗ ДАННЫХ ДЛЯ РАБОТЫ С СИСТЕМОЙ УПРАВЛЕНИЯ БАЗАМИ ДАННЫХ ПРИ ПОМОЩИ ИНСТРУМЕНТАРИЯ MYSQL DBFORGE STUDIO

*Цель работы:* ознакомиться с процессом создания базы данных в СУБД **MySQL** с использованием графического клиента **dbForge Studio**.

## ПОРЯДОК ВЫПОЛНЕНИЯ РАБОТЫ

1. Ознакомиться с теоретическими сведениями.
2. Подключится к серверу БД.
3. Изучить что хранится в служебных базах данных.
4. Создать базу данных.
5. Ознакомиться с используемыми кодировками.
6. Выполнить создание таблиц БД.
7. Изучить поддерживаемые *MySQL* подсистемы хранения таблиц.
8. Ознакомиться с используемыми в *MySQL* типами данных.
9. Выполнить настройку индексов.
10. Организовать связь между таблицами базы данных по ключевым полям.
11. Заполнить таблицы БД данными.
12. Изучить свойства связей.
13. Создать два представления *view_goods_catalog* и *view_goods_operations.*
14. Выполнить создание резервной копии базы данных.
15. Ознакомится с содержимым резервной копии БД.
16. Ответить на контрольные вопросы.

## ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Система управления базами данных **MySQL** – это продукт развития СУБД **mSQL**, которую разработчики шведской компании MySQL AB взяли за основу при создании собственной базы данных.

Достоинствами СУБД **MySQL** являются:

– высокая скорость выполнения запросов. **MySQL** наряду с **Oracle** считается одной из самых быстрых СУБД в мире;

– использование при разработке широко распространенных языков программирования C/C++;

– функционирование более чем на 20 платформах, среди которых **Windows**, **Linux**, **FreeBSD**, **HP-UX**, **Mac OS X**, **OS/2**, **Solaris** и др.;

– открытый код, который доступен для просмотра и модернизации всем желающим. Общедоступная лицензия **GPL** (General Public License) позволяет постоянно улучшать программный продукт и быстро находить и устранять уязвимые места. Особенностью лицензии **GPL** является тот факт, что любой код, скомпилированный с **GPL**-кодом, подпадает под **GPL**-лицензию, т. е. может свободно распространяться с предоставлением исходных кодов всем желающим;

– поддержка программного интерфейса приложения **API** (*Application Programming Interface* – программный интерфейс приложения) для огромного количества языков программирования;

– наличие встроенного сервера. СУБД **MySQL** может использоваться в качестве как внешнего сервера, поддерживающего соединение с локальным компьютером и с удаленным хостом, так и встроенного сервера. Достаточно скомпилировать программу с библиотекой встроенного сервера и приложение уже будет содержать полноценную СУБД **MySQL** с возможностью создания баз данных, таблиц и осуществления запросов к ним;

– широкий выбор подсистем хранения таблиц, в том числе и сторонних разработчиков, что позволяет реализовать оптимальные для решаемой задачи производительность и функциональность;

– корректно выполненная локализация. Например, у пользователя не возникнет проблем из-за того, что в конце работы над проектом поддержка базой данных сортировки русских строк окажется невозможной;

– начиная с версии 5.0, практически полное соответствие стандарту языка SQL и, следовательно, совместимость с другими базами данных.

Все эти особенности сделали **MySQL** стандартной базой данных, используемой на серверах хост-провайдеров и в качестве встроенной базы данных в прикладных программах.

Далее мы рассмотрим вопросы, касающиеся создания и администрирования базы данных в СУБД **MySQL**.

## 1. Создание базы данных в программе dbForge Studio for MySQL

Для создания новой пустой базы данных будем использовать визуальный редактор **dbForge Studio** (существуют и другие клиентские приложения для работы с базами данных MySQL: *HeidiSQL, MySQL Workbench, DBeaver, PHPMyAdmin* и другие. *dbForge Studio* используется так как является наиболее функциональной системой).

Базу данных для **MySQL** можно формировать в консольном режиме с использованием **SQL**-команд через клиента **mysql.exe**, который поставляется вместе с сервером, но такой способ является трудоемким, поскольку в нашем случае для создания базы данных, таблиц и связей между ними большое количество **SQL**-конструкций придется набирать вручную.

### 1.1. Подключение к серверу БД

Необходимо установить соединение с запущенным сервером **MySQL**, который может находиться как на локальном компьютере, так и удаленно.

Выберем пункт меню **База данных** и в открывшемся списке щелкнем на пункте **Новое соединение** (рис. 1.1). В качестве хоста зададим **localhost**. *MySQL* по умолчанию использует порт **3306**. Работа с базой данных будет осуществляться от имени суперпользователя. Для этого в поле **Имя** запишем **root**. Дадим название соединению и нажмем кнопку **Соединить**.

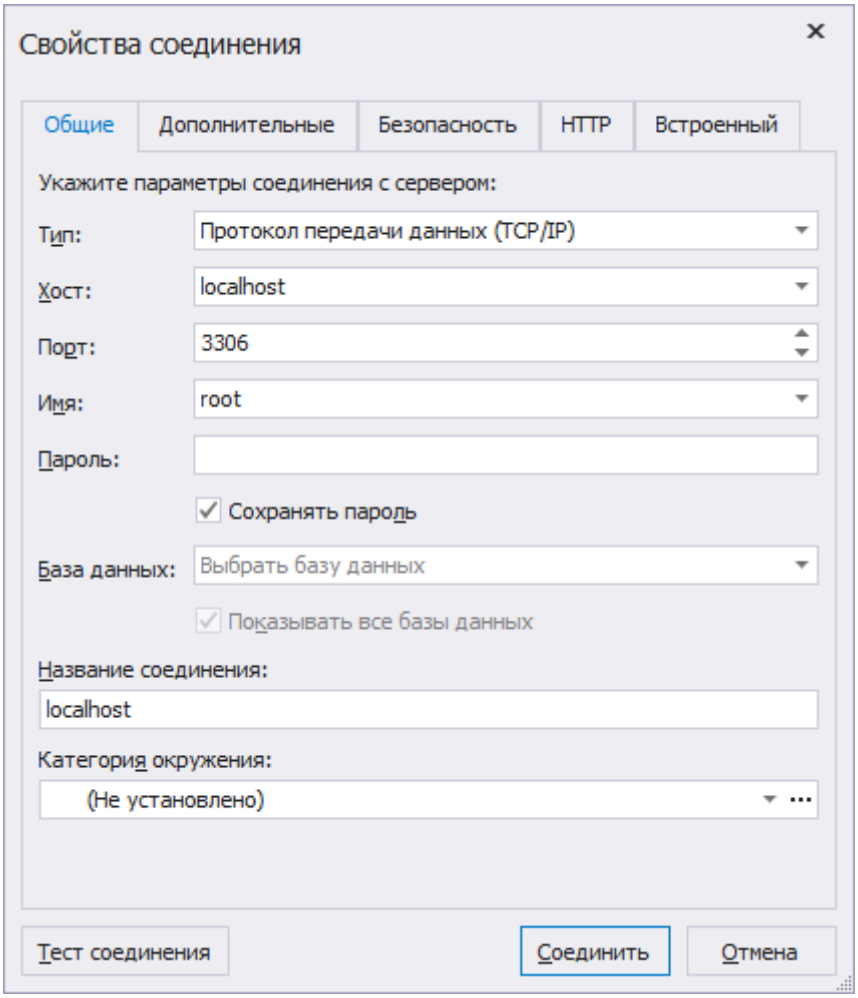


Рис. 1.1. Создание нового соединения

При соединении с сервером в области проводника будут отображены служебные базы данных **information_schema**, **mysql**, **perfomation_schema**, необходимые для функционирования и администрирования СУБД.

База данных **information_schema** обеспечивает доступ к метаданным базы данных и является набором доступных только для чтения представлений (**views**), которые содержат информацию обо всех таблицах, представлениях, столбцах и процедурах, хранящихся в БД.

Служебная база данных **mysql** является системным каталогом СУБД, создается во время установки **MySQL** и хранится на диске. Она содержит информацию, необходимую для правильного функционирования СУБД: о поддерживаемых базах данных и их базовых таблицах, представлениях, курсорах, индексах, пользователях и их правах доступа к информации и т. д. Пользователь, имеющий права администратора, может модифицировать, добавлять и удалять данные из системного каталога.

База данных **performance_schema** предназначена для низкоуровневого мониторинга производительности сервера БД. Она позволяет пользователям контролировать наиболее ресурсоемкие запросы, объекты, приложения и учетные записи, обеспечивает доступ к сводкам с агрегированной статистикой.

## 1.2. Создание БД

Далее создадим собственную базу данных, выбрав пункт меню **База данных** и щелкнув на подпункте меню **Новая база данных** (рис. 1.2).

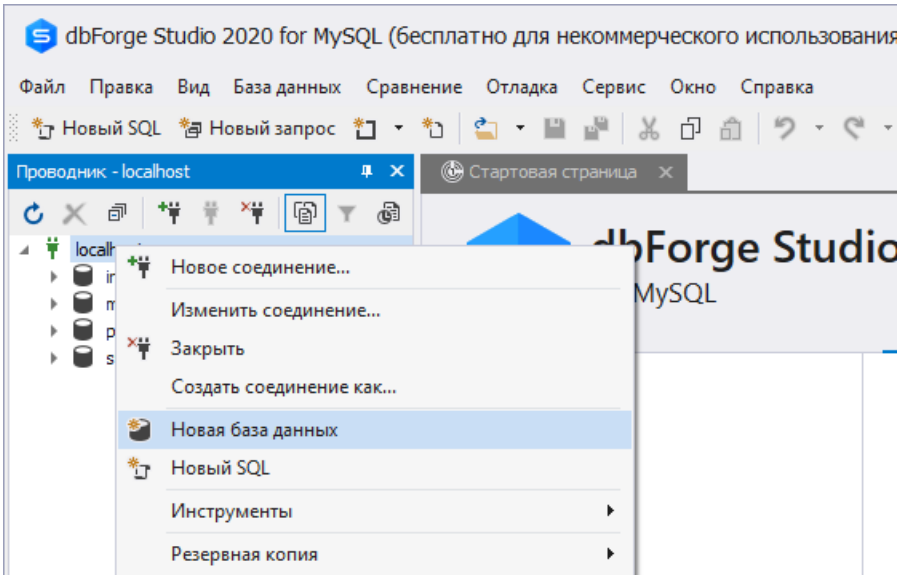



Рис. 1.2. Пример создания БД

Введем имя базы данных **GoodsManager** (БД будет представлять из себя магазин компьютерной техники) выберем кодировку **utf8mb4** (*UTF*-8 *Multibyte* 4) и нажмем кнопку применить изменения (рис. 1.3). Кодировка - это таблица сопоставлений символов, которые мы можем видеть на экране, определенным числовым кодам. Т.е. каждый символ, который мы вводим с клавиатуры, либо видим на экране монитора, закодирован определенной последовательностью битов (нулей и единиц).

Внешний вид самих символов определяется файлами шрифтов, которые установлены на вашем компьютере.

В контексте разработки, можно столкнуться с некорректным отображением символов, когда, к примеру, русский текст в базе хра-

нится в одной кодировке (*UTF*-8), а в разрабатываемом приложении по ошибке отображается с помощью другой (*Windows*-1251).

Решить эту проблему позволяет применение универсальной кодировки **UTF-8** (*Unicode Transformation Format*) – это многобайтовая кодировка с переменной длинной символа. В своей базе мы применяем расширенный вариант **utf8mb4**, которая позволяет кодировать дополнительные символы, например, *Emoji*.

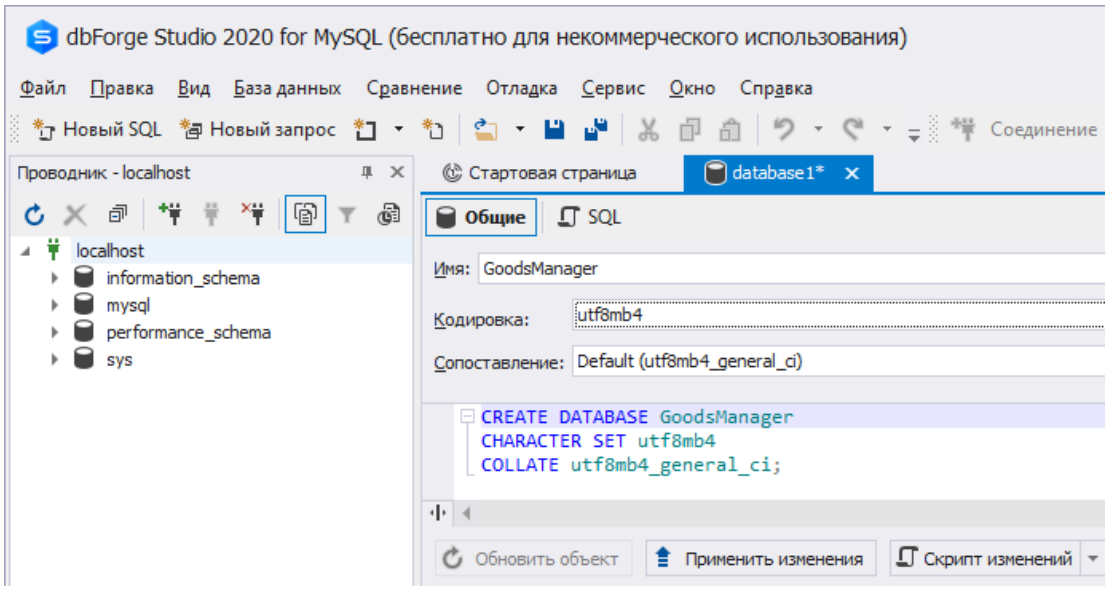



Рис. 1.3. Пример настройки БД

Так как мы работаем с серверной системой, которая обычно находится на другом компьютере, необходимо обновлять информацию после выполнения каких-либо действий с БД (все запросы выполняются на сервере, и чтобы увидеть применяемые изменения в приложении, необходимо чтобы сервер отправил нам их) (рис. 1.4).

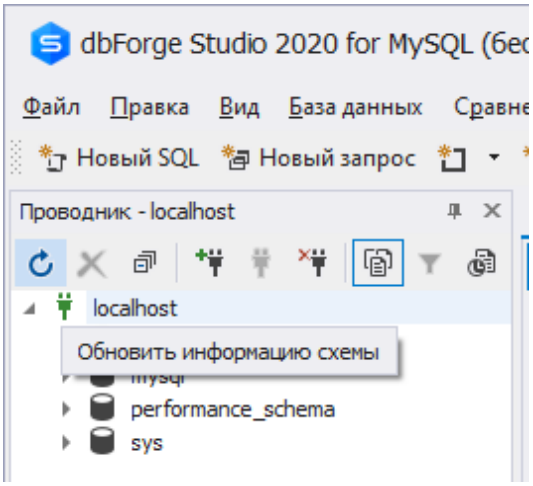



Рис. 1.4. Пример обновления данных

## 1.3. Создание таблиц

Для создания таблиц выберем базу данных в проводнике, щелкнем правой кнопкой мыши на подкаталоге **Таблицы** и в появившемся контекстном выберем пункт **Новая таблица** (рис. 1.5).

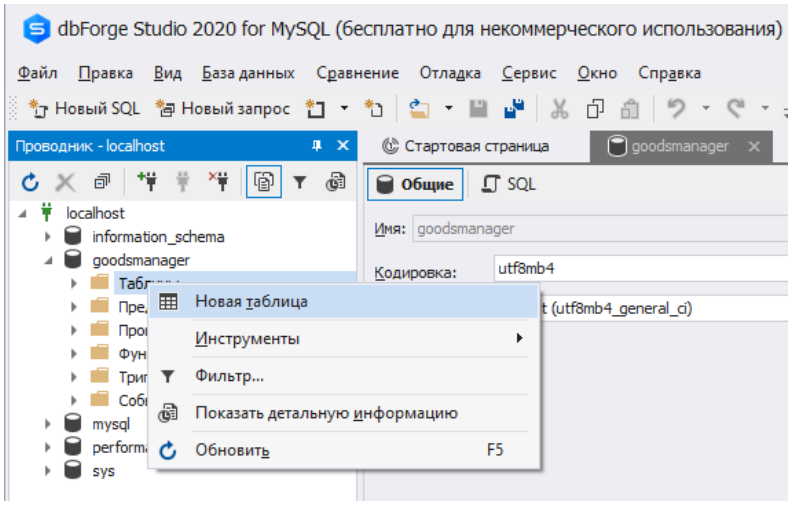



Рис. 1.5. Создание таблицы

Введем имя таблицы **type_goods** (Категории товаров), укажем подсистему хранения (**Движок**, *Engine*), как **InnoDB** и создадим новый столбец **IDtg** (Код типа товаров – идентификатор, по которому можно ссылаться на конкретную категорию товаров) (рис. 1.6).

Укажем, что **IDtg** является ключевым (*PRIMARY KEY*) полем установив флажок напротив имени столбца.

Для каждого создаваемого поля таблицы (рис. 1.6) выбирается его тип и длина (точность и масштаб). Точность (*Precision*) представляет собой общее количество цифр в числе, а масштаб (*Scale*) задает количество десятичных знаков после запятой. Например, число **123.45** имеет точность **5** и масштаб **2**.

С помощью флажков выставляются свойства полей: беззнаковое, автоинкремент (для ключей), пустые значения, уникальность и др.

Атрибут **Автоинкремент** (*AUTO_INCREMENT*) используется для генерации уникальных идентификаторов для новых строк таблицы.

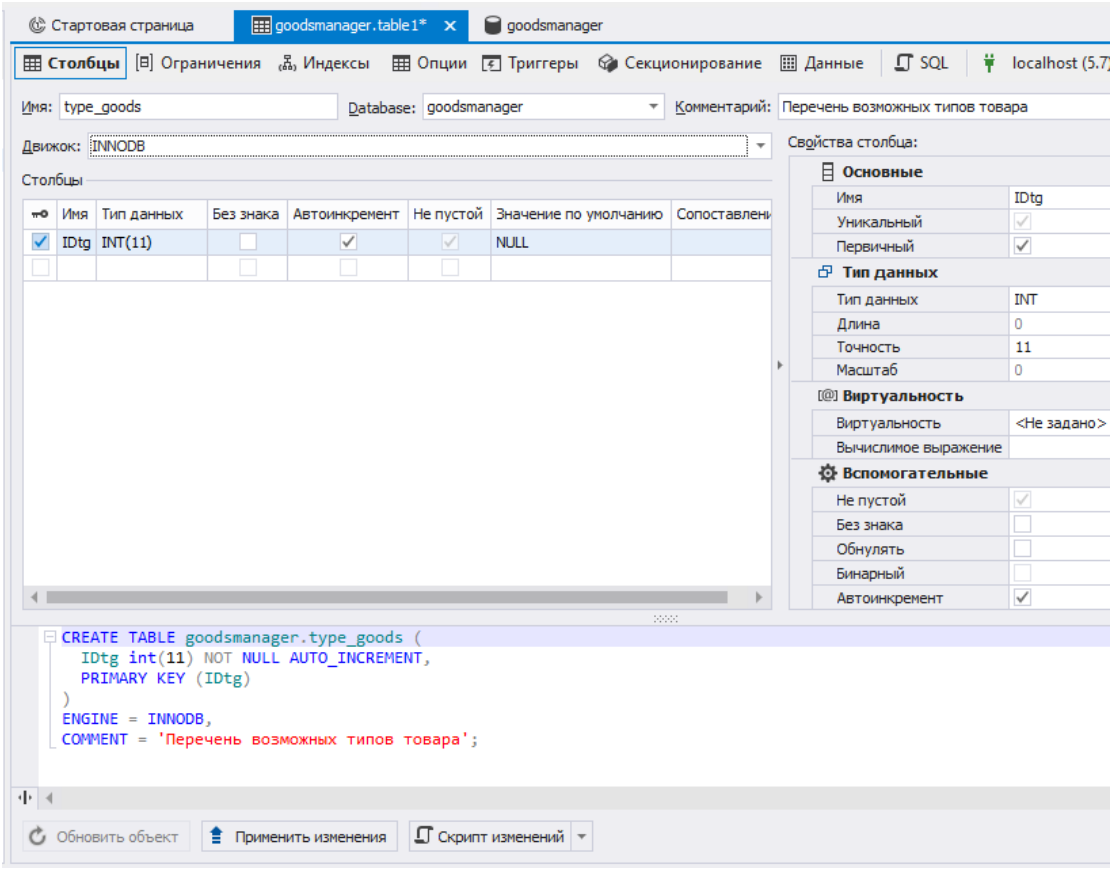



Рис. 1.6. Создание столбцов таблицы

*MySQL* поддерживает различные подсистемы хранения таблиц (*Engines*):

– **InnoDB** наиболее широко используемый движок с поддержкой транзакций (*ACID*). Основное отличие от других, поддержка ссылочной целостности с внешними ключами.

– **MyISAM** применяется для не транзакционных таблиц с блокировками на уровне таблицы и полнотекстовым поиском для индексов. В основном используется в *web*.

– **Federated** предоставляет возможность объединять различные *MySQL*-серверы для создания одной логической базы данных из нескольких физических компьютеров.

– **Memory** создает таблицы и хранит их в оперативной памяти для повышения быстродействия и ускорения доступа к данным.

– **Merge** используется для логического объединения одинаковых *MyISAM*-таблиц в одну. Может использоваться для обработки больших объемов данных.

– **Archive** используется для хранения больших объемов неиндексированных данных.

– **CSV** гибкое хранилище для CSV (*Comma-Separated Values*) файлов. Используется для экспорта/импорта данных в этом формате.

– **Blackhole** принимает данные, но ничего не возвращает.

*Таблица 1.1*

**Строковые типы данных**

| **Тип данных** | **Описание** |
|---|---|
| CHAR(size) | Строка **фиксированной** длины (может содержать буквы, числа, специальные символы), *size* - количество символов 0 до 255 |
| VARCHAR(size) | Строка **переменной** длины (может содержать буквы, числа, специальные символы), *size* - количество символов 0 до 65535 |
| BINARY(size) | Соответствует CHAR, но хранит строки в бинарном виде. *size* – длина строки в байтах |
| VARBINARY(size) | Соответствует VARCHAR, но хранит строки в бинарном виде, *size* – длина строки в байтах |
| TINYBLOB | Для бинарных объектов (Binary Large Objects). Максимальная длина: 255 байт |
| TINYTEXT | Хранит строки длиной до 255 символов |
| TEXT(size) | Хранит строки длиной до 65,535 байт |
| BLOB(size) | Для бинарных объектов. Хранит до 65,535 байт данных |
| MEDIUMTEXT | Хранит строки длиной до 16,777,215 символов |
| MEDIUMBLOB | Для бинарных объектов. Хранит до 16,777,215 байт данных |
| LONGTEXT | Хранит строки длиной до 4,294,967,295 символов |

| | |
|---|---|
| LONGBLOB | Для бинарных объектов. Хранит до 4,294,967,295 байт данных |
| ENUM(v1, v2, v3, ...) | Строковый объект который имеет только одно значение, выбираемое из списка возможных значений. Список может содержать 65535 значений. |
| SET(v1, v2, v3, ...) | Строковый объект который имеет 0 или более значений, выбираемое из списка возможных значений. Список может содержать 64 значения |

*Таблица 1.2*

**Числовые типы данных**

| **Тип данных** | **Описание** |
|---|---|
| BIT(*size*) | Битовое значение. Количество бит на хранимое данное, которое задается *size* и может быть от 1 до 64 |
| TINYINT(*size*) | Маленькое целое число в диапазоне от -128 до 127, если со знаком. Без знака от 0 до 255, *size* количество отображаемых значений (до 255) |
| BOOL | Ноль считается как false, остальные значения как true |
| BOOLEAN | Соответствует BOOL |
| SMALLINT(*size*) | Небольшое целое число в диапазоне от -32768 до 32767, если со знаком. Без знака от 0 до 65535, *size* количество отображаемых значений (до 255) |
| MEDIUMINT(*size*) | Среднее целое число в диапазоне от -8388608 до 8388607, если со знаком. Без знака от 0 to 16777215, *size* количество отображаемых значений (до 255) |
| INT(*size*) | Целое число в диапазоне от -2147483648 до 2147483647, если со знаком. Без знака от 0 до 4294967295 |

| | |
|---|---|
| INTEGER(*size*) | Соответствует INT |
| BIGINT(*size*) | Большое целое число в диапазоне от -223372036854775808 до 9223372036854775807, если со знаком. Без знака от 0 до 18446744073709551615 |
| FLOAT(*size*, *d*) | Число с плавающей точкой. Общее количество цифр определяется *size*. Количество цифр после запятой определяется *d*. |
| DOUBLE (*size*, *d*) | Число с плавающей точкой двойной точности. Общее количество цифр определяется *size*. Количество цифр после запятой определяется *d*. |
| DECIMAL(*size*, *d*) | Число с **фиксированной** точкой. Общее количество цифр определяется *size*. Количество цифр после запятой определяется *d*. Максимум для *size* - 65. Максимум для *d* - 30. По умолчанию *size* равен 10, а *d* равно 0. |
| DEC(*size*, *d*) | Соответствует to DECIMAL |

*Таблица 1.3*

**Типы данных для хранения даты и времени**

| **Типа данных** | **Описание** |
|---|---|
| DATE | Дата в формате: YYYY-MM-DD. Поддерживает диапазон дат от '1000-01-01' до '9999-12-31' |
| DATETIME(*fsp*) | Комбинация даты и времени в формате: YYYY-MM-DD hh:mm:ss. Поддерживает диапазон дат от '1000-01-01 00:00:00' до '9999-12-31 23:59:59'. По умолчанию инициализируется текущей датой и временем. |
| TIMESTAMP(*fsp*) | Временная метка. Значения хранятся как количество секунд, прошедшее с '1970-01-01 00:00:00' UTC (Unix epoch). Формат: YYYY-MM-DD hh:mm:ss. Поддерживает диапазон дат от '1970-01-01 00:00:01' UTC до '2038-01-09 03:14:07' UTC. |

| | |
|---|---|
| TIME(*fsp*) | Время в формате: hh:mm:ss. Поддерживает диапазон от '-838:59:59' до '838:59:59' |
| YEAR | Год в формате четырех цифр. Доступные значения: от 1901 до 2155, и 0000 |

Далее создадим новый столбец **tgName** (Наименование категории товаров) (рис. 1.7). **Комментарии** (COMMENT) не обязательны, но вводятся для того, чтобы разработчикам БД было в будущем легче сориентироваться предметной области.

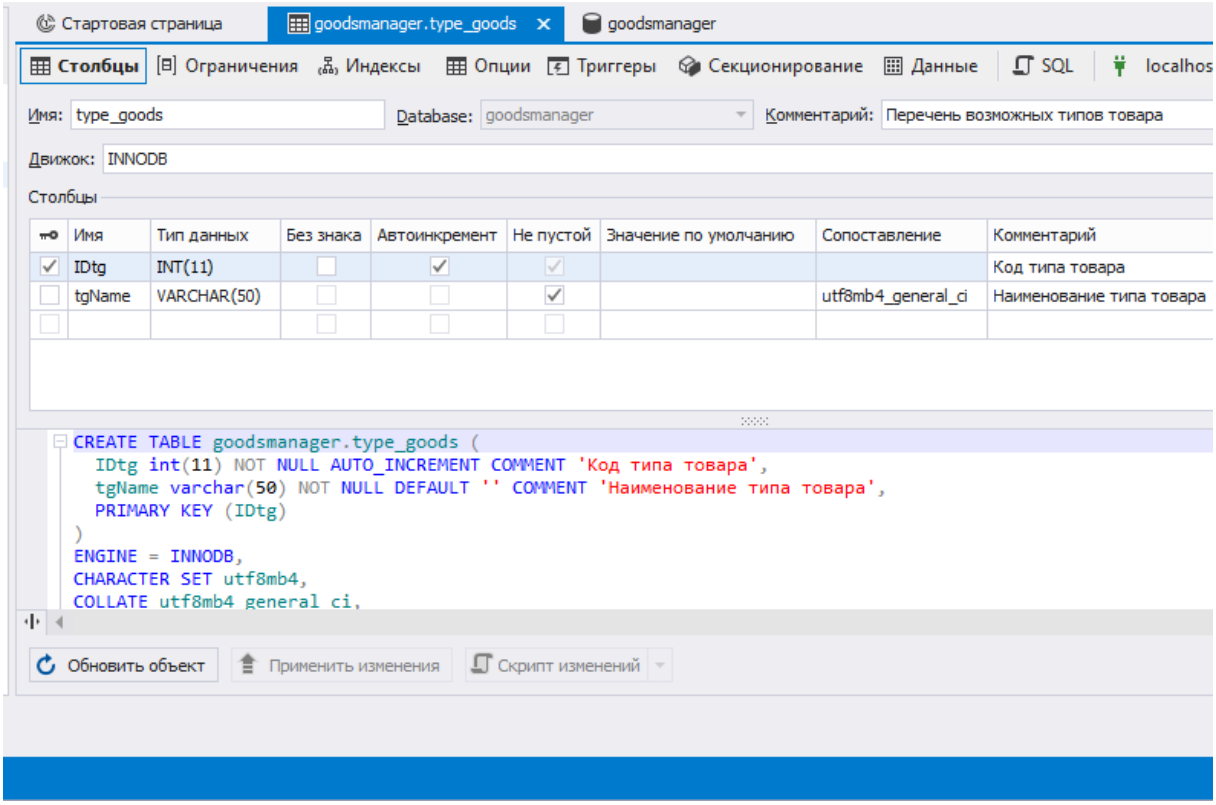



Рис. 1.7. Создание столбца *tgName*

Перейдем к созданию индексов для таблицы **type_goods**.

Индексы – это вспомогательные объекты БД предназначенные для повышения производительности поиска. Индекс создается для одного или нескольких полей (составной индекс). Без индекса чтение данных осуществляется по всей таблице, начиная с первой записи, пока не будут найдены соответствующие строки. В этом случае, чем больше таблица, тем больше будут накладные расходы.

Перейдите в раздел **Индексы** (*Indexes*) и создайте уникальный индекс для столбца **tgName** щелкнув правой кнопкой на пустую область (уникальность индекса будет контролировать, чтобы вводимые категории товаров не повторялись) (рис. 1.8). Примените изменения.

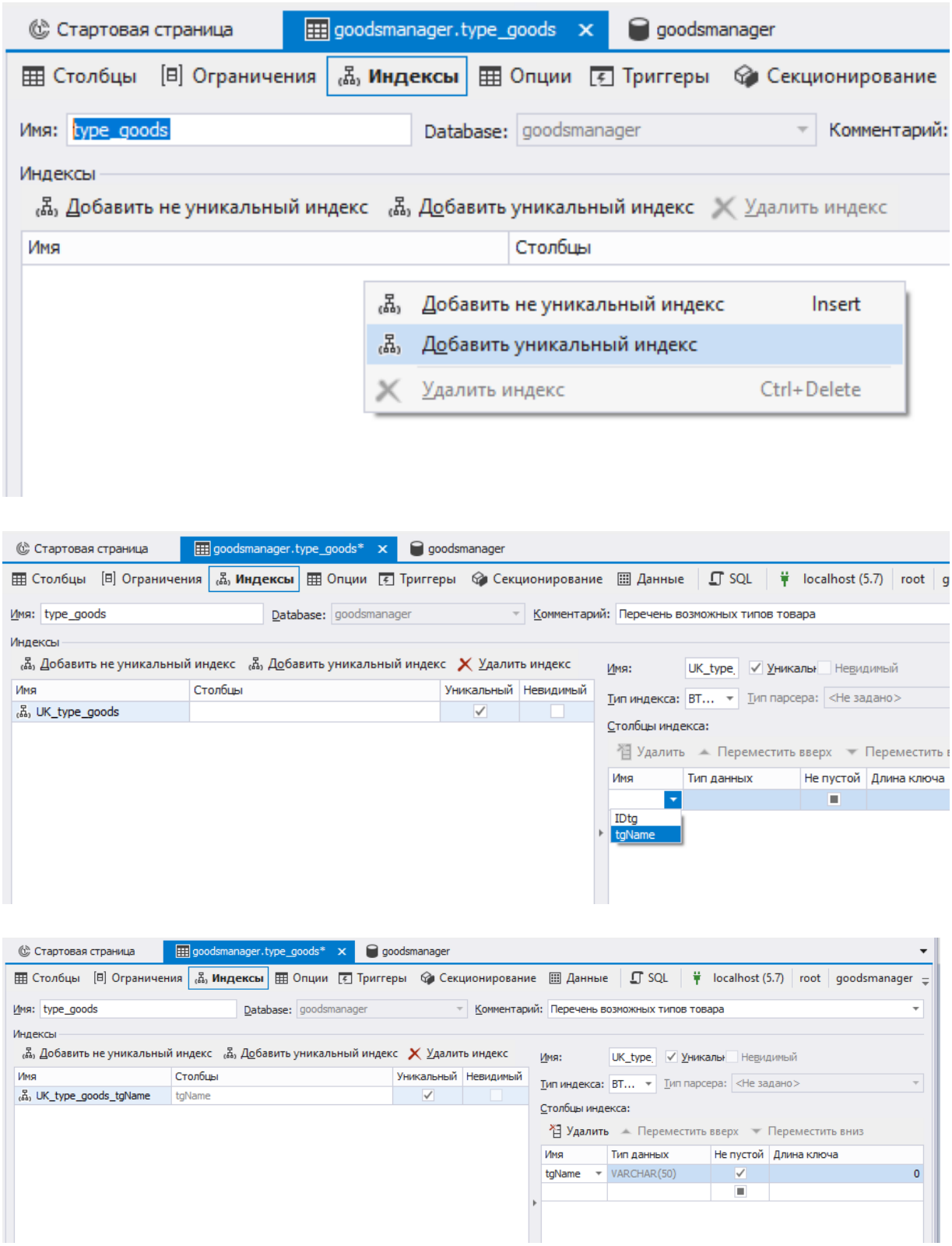



Рис. 1.8. Создание уникального индекса

По умолчанию используется тип индекса **B-дерево** (*B-tree*), который покрывает широкий круг задач. С помощью B-дерева можно проиндексировать любые данные, которые могут быть отсортированы, т. е. для которых применимы операции сравнения больше/меньше/равно (числа, строки, даты и время, логический тип и любые данные, которые можно ими закодировать).

Индексы важный инструмент баз данных, ускоряющий поиск, но не стоит создавать много индексов без необходимости, так как они занимают дополнительную память, и при любом обновлении проиндексированных данных СУБД необходимо перестроить и сам индекс.

Далее необходимо добавить записи в таблицу. Перейдите в раздел **Данные**, выберите необходимую таблицу и добавьте новую запись нажав клавишу **Insert** (рис. 1.9). Далее нажмите **Запомнить**.

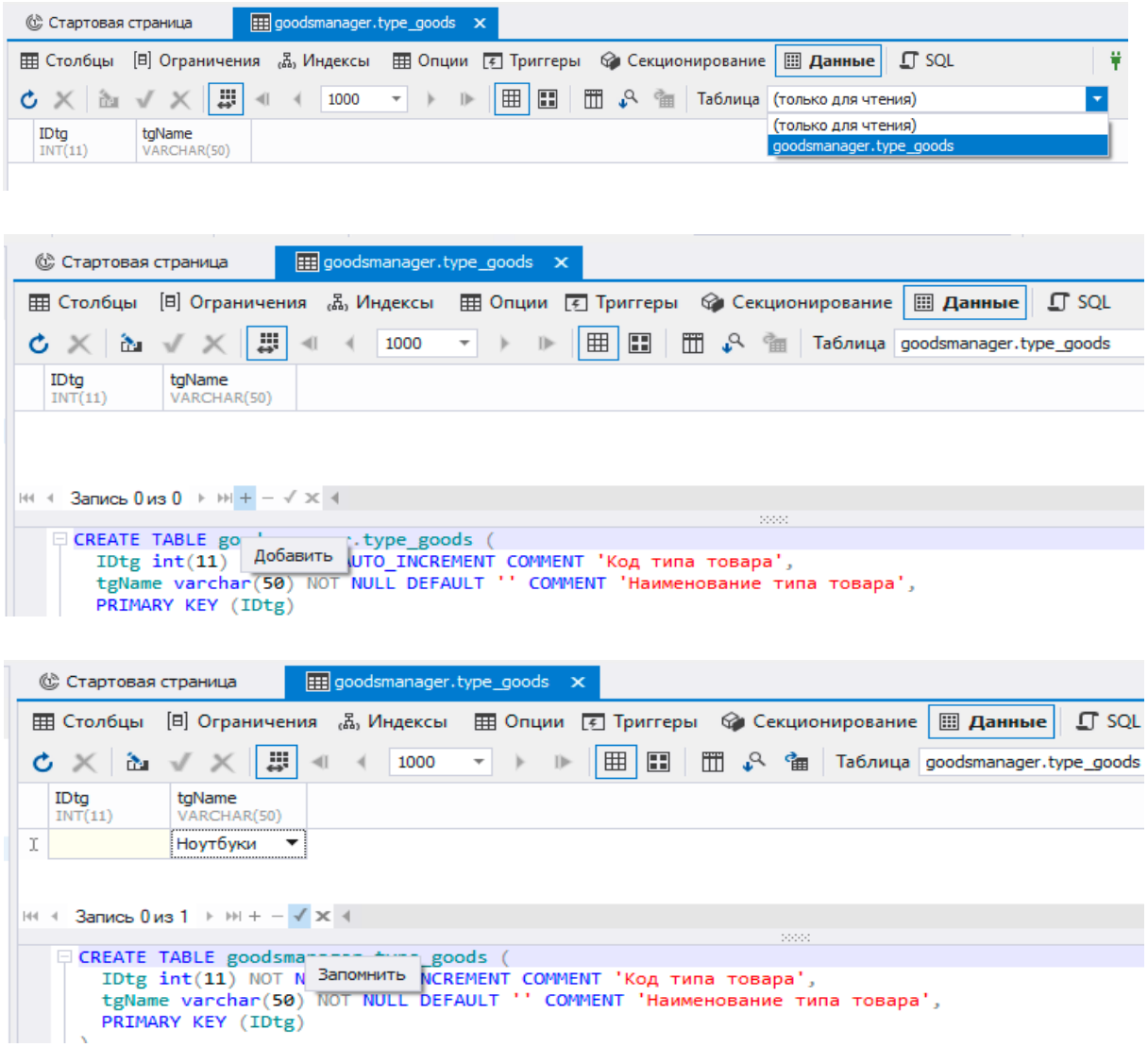



Рис. 1.9. Добавление записи в таблицу

В БД магазина будет всего три категории товаров (рис. 1.10).

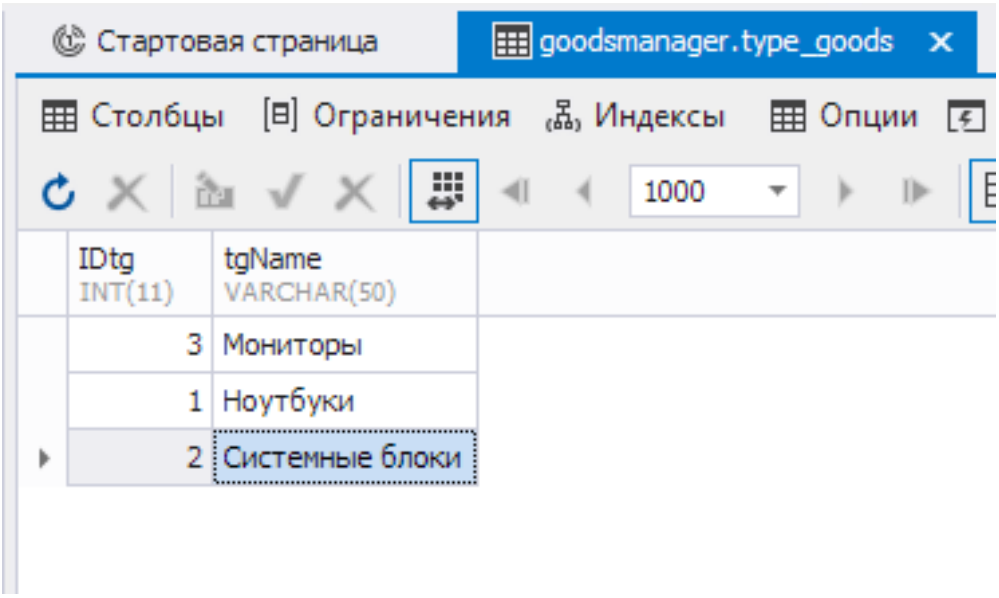



Рис. 1.10. Заполненная таблицы *type_goods*

Создайте еще одну таблицу **goods_catalog** для хранения конкретных товаров определенной категории (рис. 1.11). Примените изменения нажав **Применить изменения**.

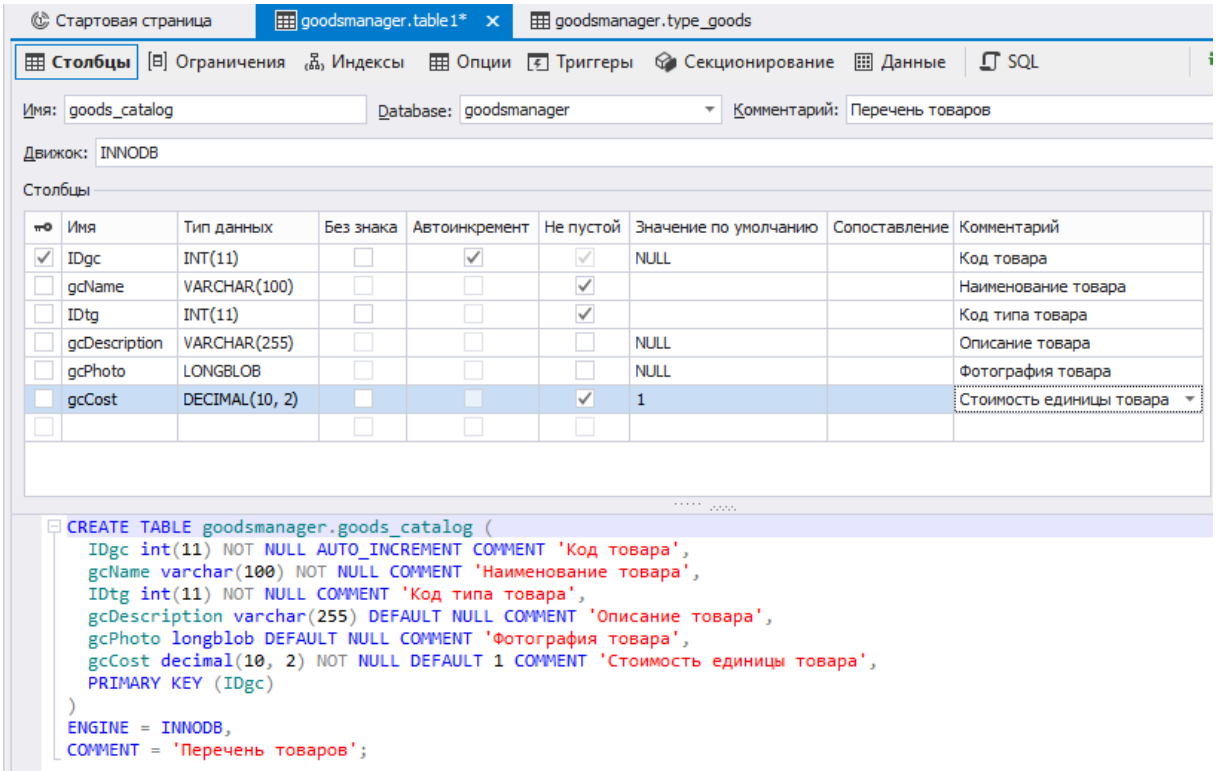



Рис. 1.11. Создание таблицы *goods_catalog*

Далее необходимо обеспечить ссылочную целостность данных создав связь между категорией и самим товаром. Перейдите в раздел **Ограничения** и добавьте внешний ключ (*FOREIGN KEY*) кликнув правой кнопкой на пустое место и выбрав **Создать внешний ключ** (рис. 1.12).

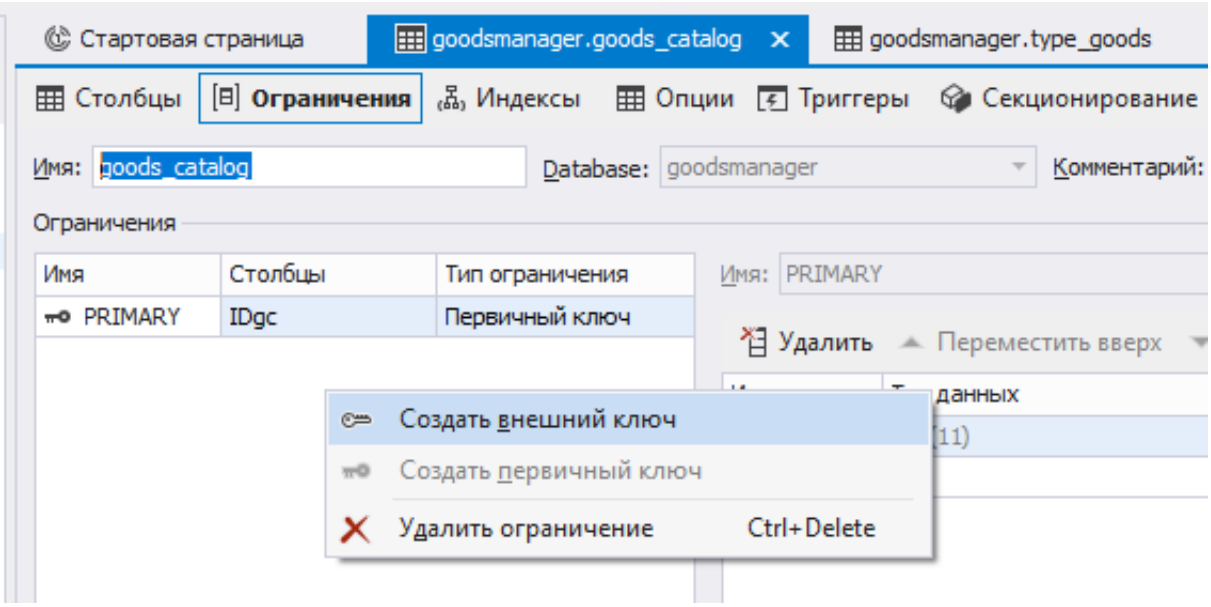
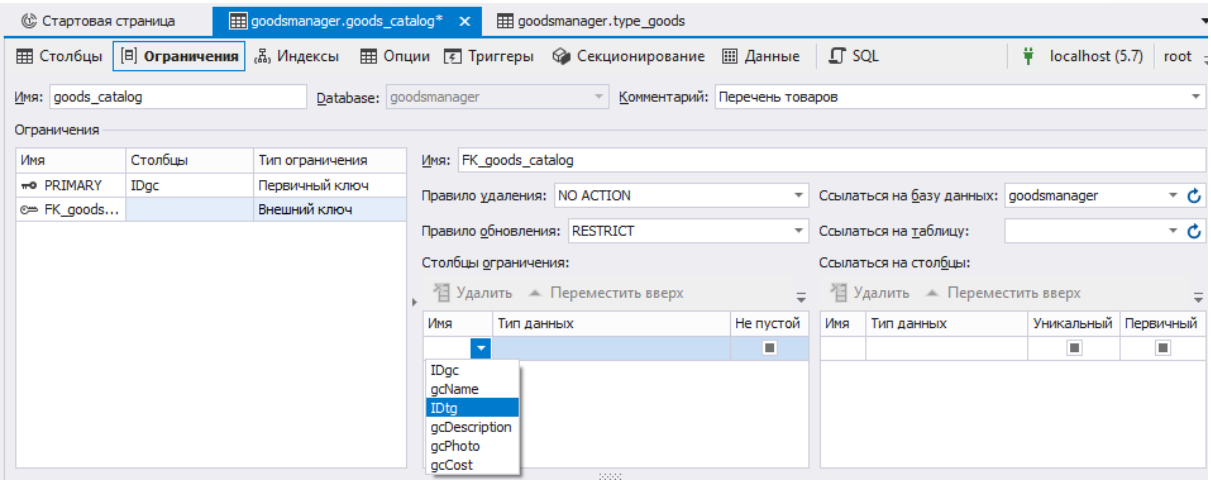



Рис. 1.12. Создание внешнего ключа

Внешний ключ должен связывать поля **IDtg** в обоих таблицах, являющихся идентификаторами категории товаров (рис. 1.13). Внешние ключи создаются для **дочерних** таблиц. Дочерними считаются таблицы которые зависят от других таблиц, например, товары из *goods_catalog* ссылаются на категории товаров из *type_goods.*



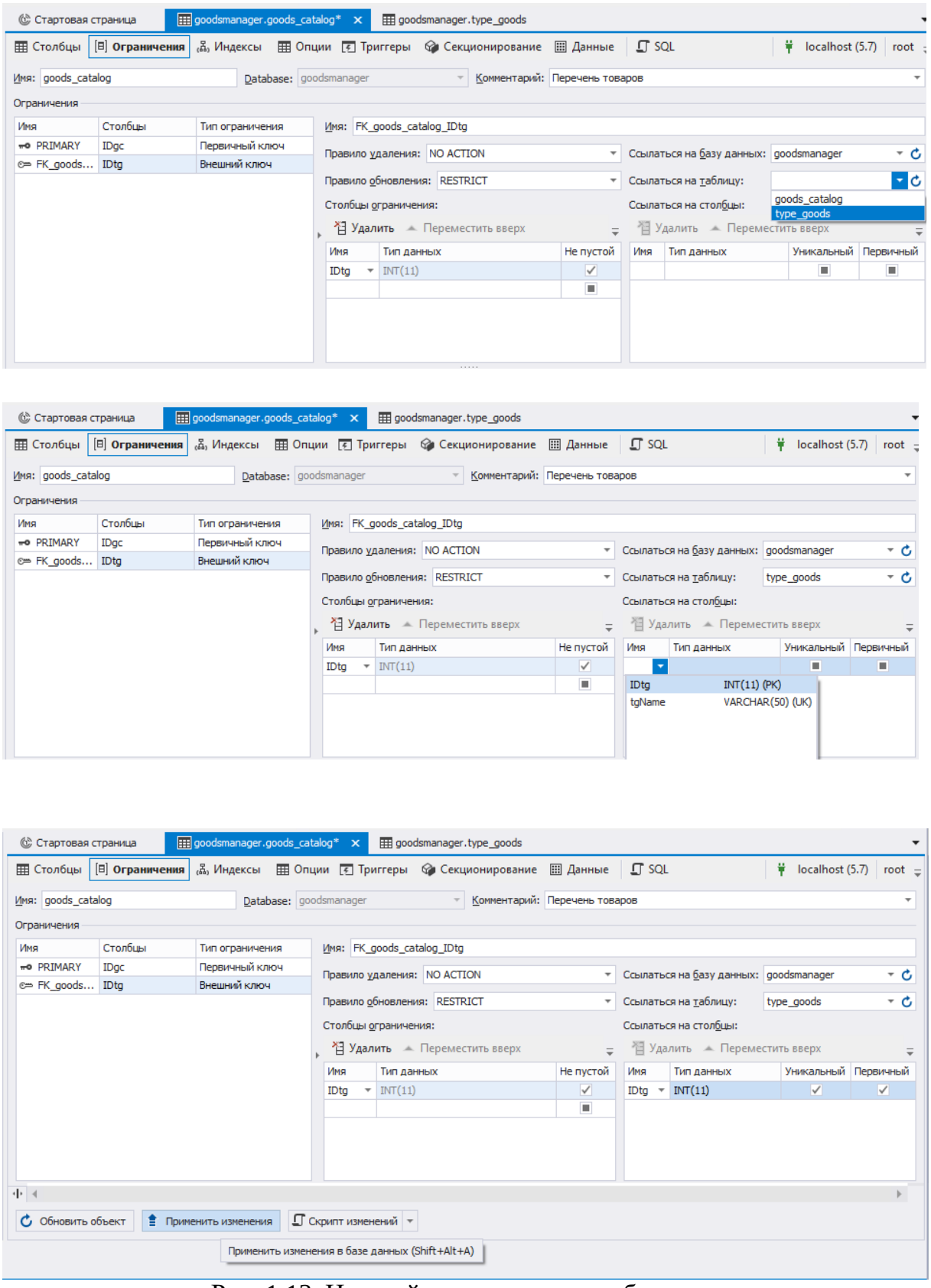



Рис. 1.13. Настройка связи между таблицами

Внешний ключ – это столбец или сочетание столбцов, которое применяется для принудительного установления связи между данны-

ми в двух таблицах. Для внешнего ключа можно настроить правила обновления и удаления данных:

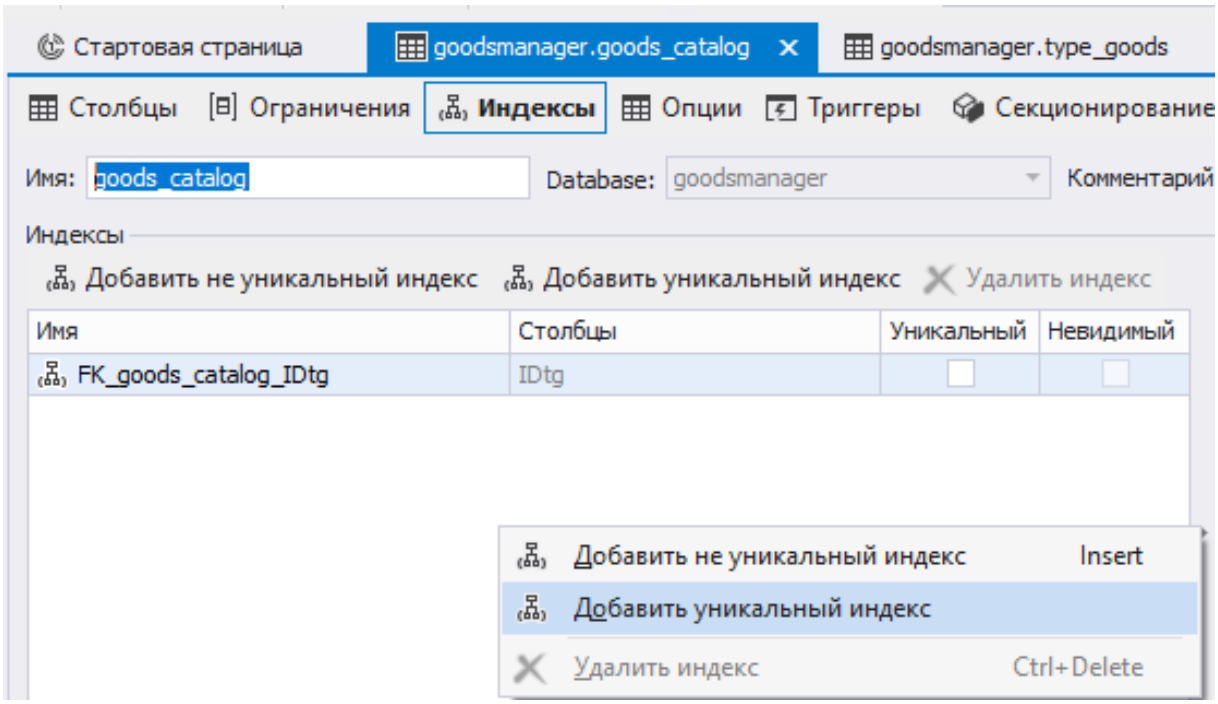
– **CASCADE** – удаление или изменение записи, содержащей первичный ключ, приведет к автоматическому удалению или изменению соответствующих записей в дочерней таблице. Например, удаляя категорию **Ноутбуки** в *type_goods* также удалятся все товары из *goods_catalog* относящиеся к этой категории;

– **SET NULL** – при удалении или изменении записи, содержащей первичный ключ, соответствующие записи в дочерней таблице получат значение **NULL**;

– **RESTRICT** – нельзя удалить или изменить запись, пока в дочерней таблице существуют ссылающиеся записи;

– **NO ACTION** – в *MySQL* это правило работает аналогично *RESTRICT*.

Добавим несколько индексов в таблицу **goods_catalog** (рис. 1.14). Один будет уникальным и создан для столбца **gcName**, второй будет не уникальным и составным, созданный для полей **IDtg** и **gcName**.



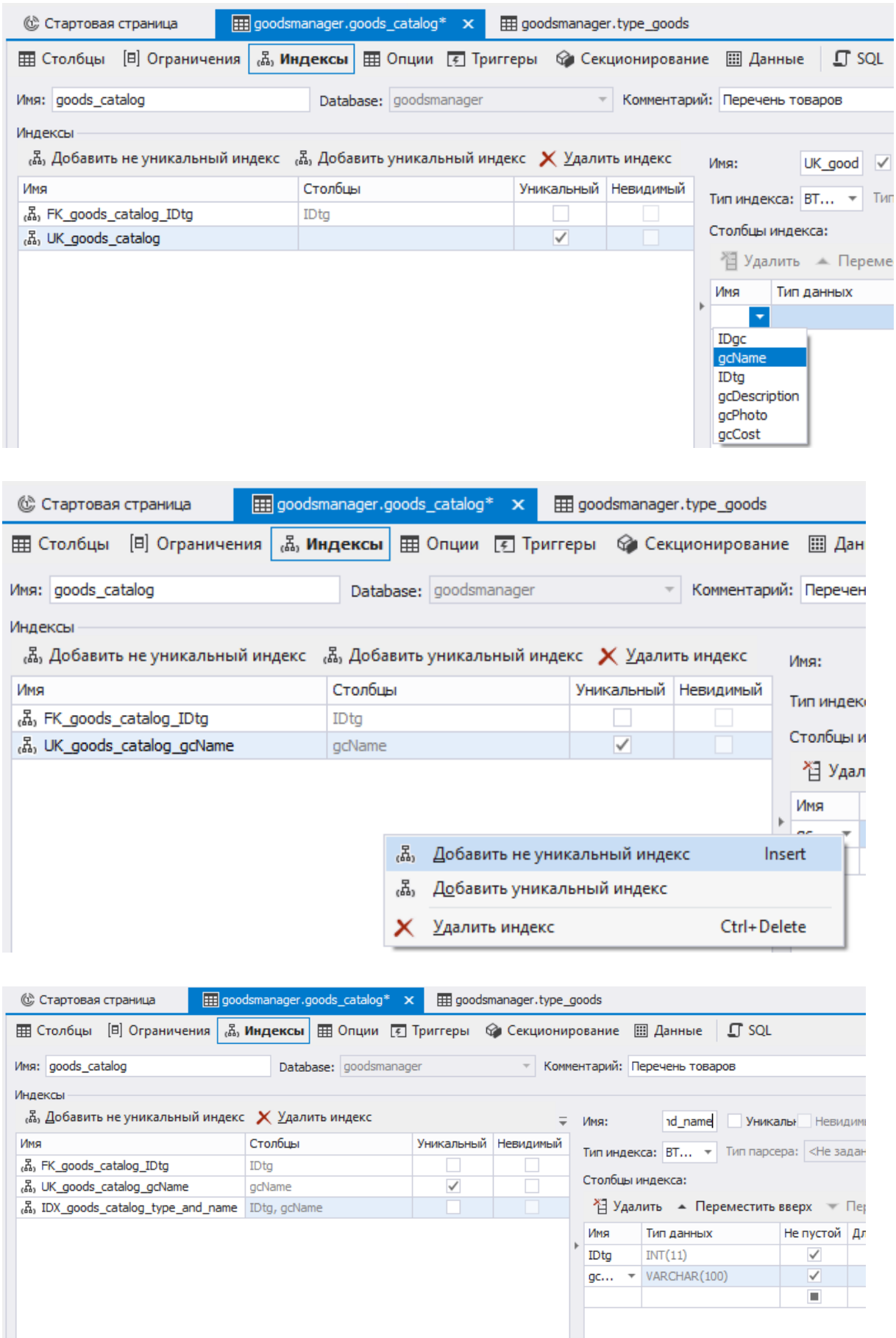
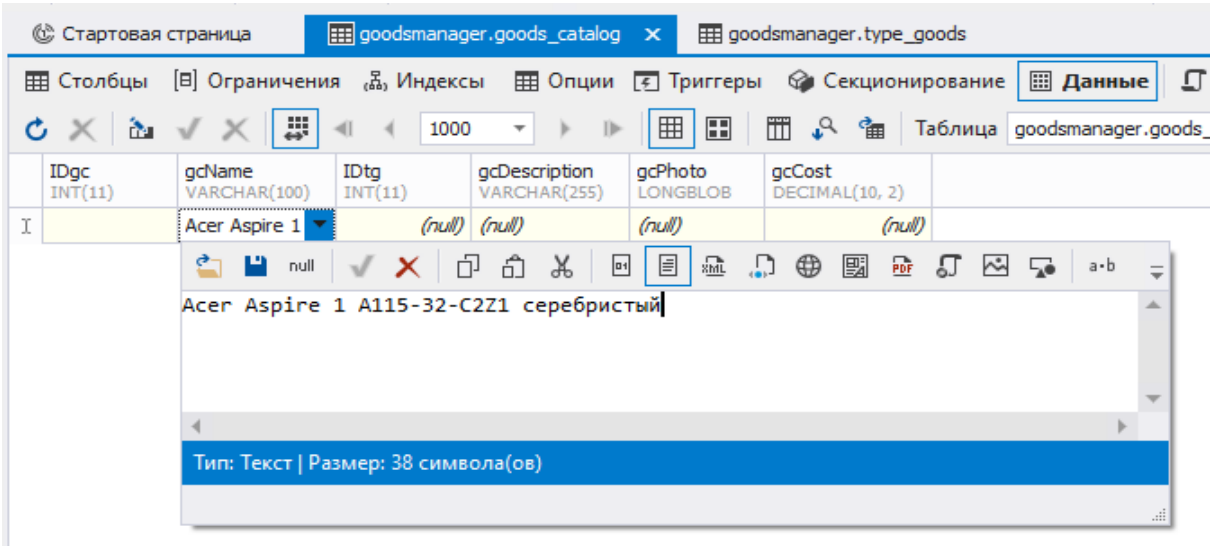
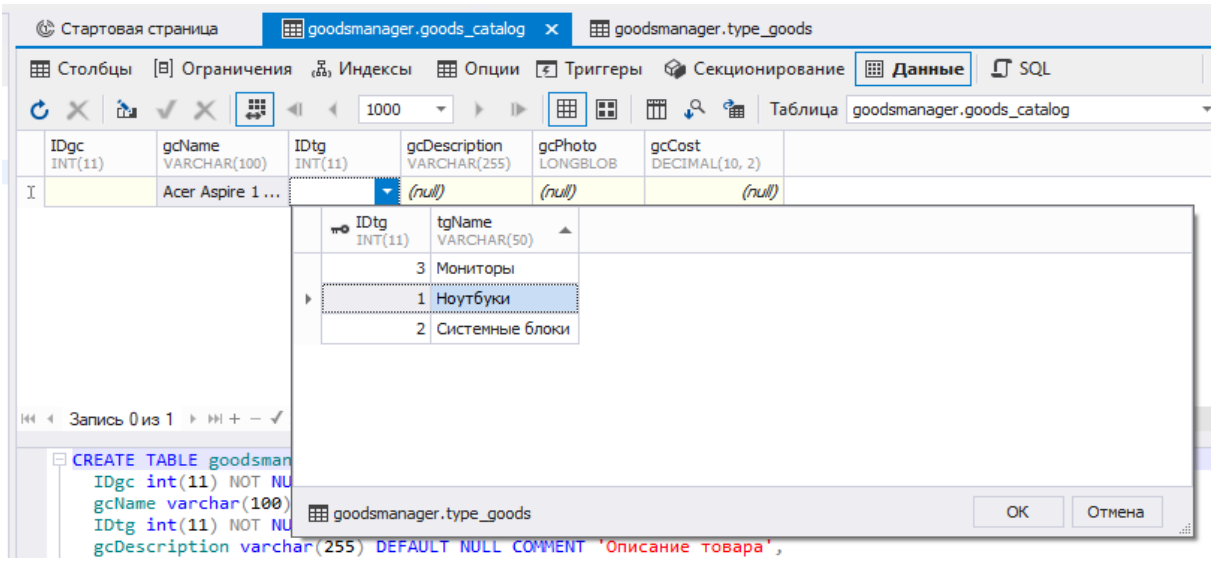
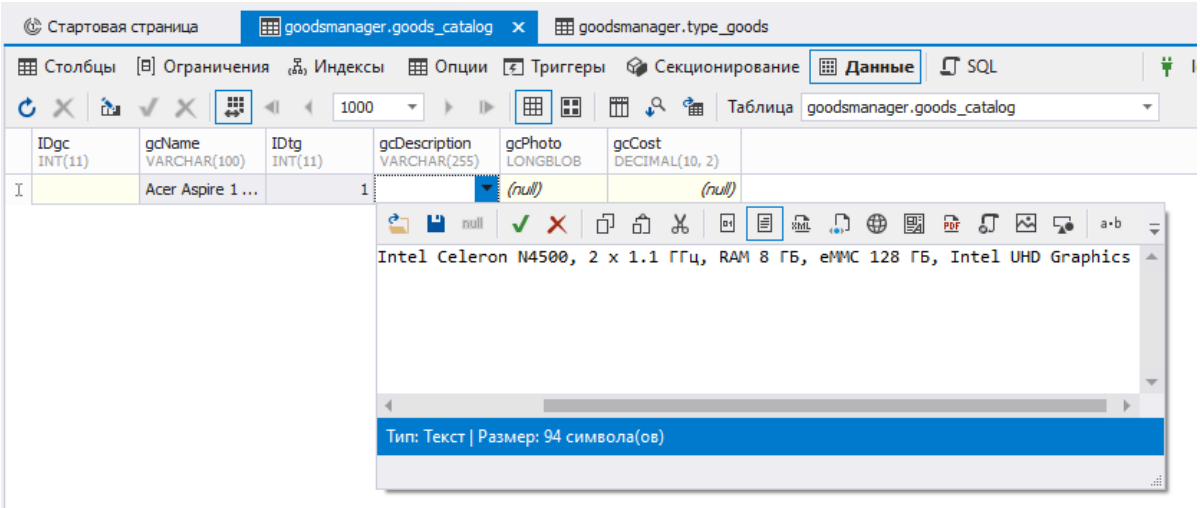



Рис. 1.14. Настройка индексов для *goods_catalog*

Заполните таблицу **goods_catalog** данными (рис. 1.15).



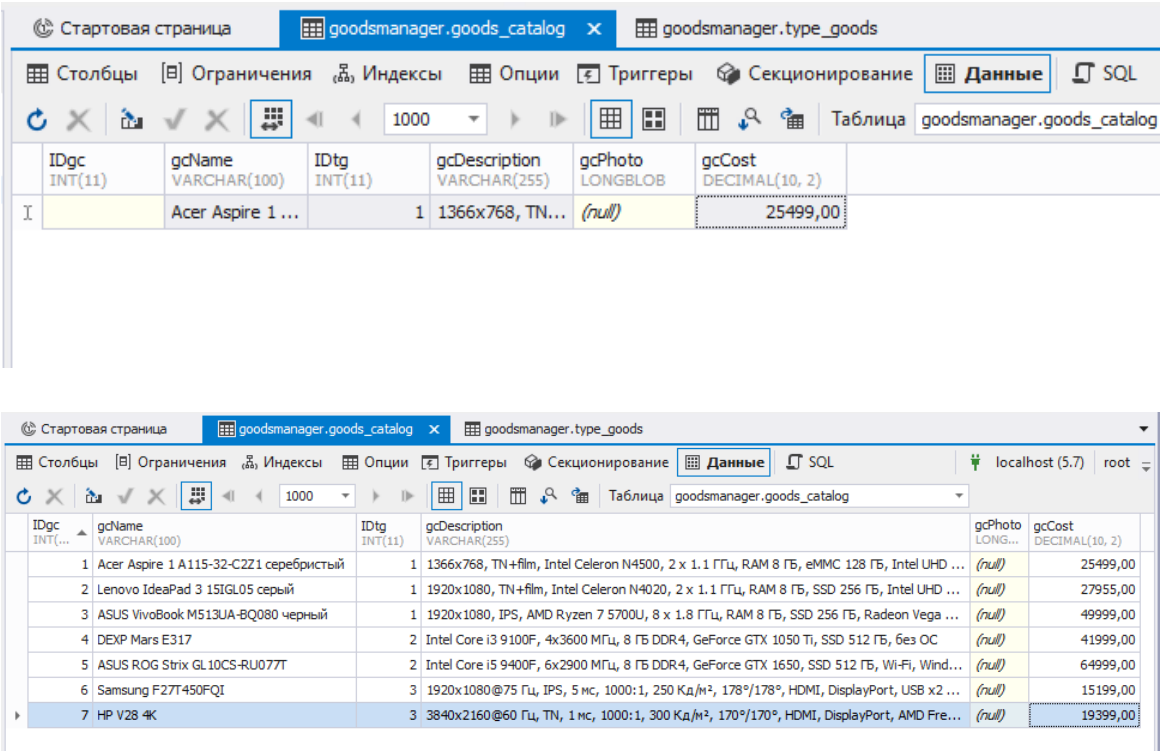



Рис. 1.15. Заполнение данными таблицы *goods_catalog*

Создадим таблицу **suppliers_catalog**, хранящую данные о поставщиках товаров (рис. 1.16).

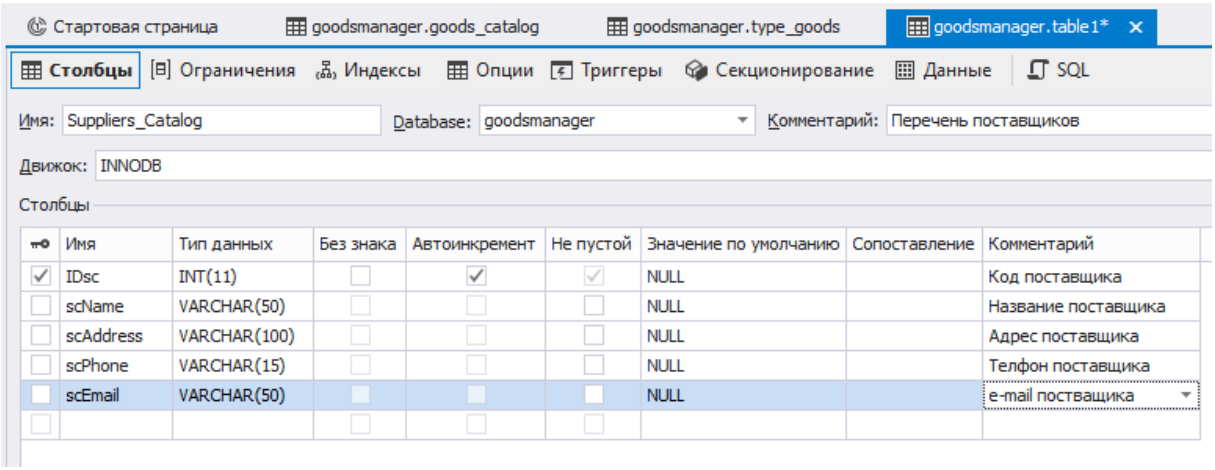



Рис. 1.16. Создание таблицы *suppliers_catalog*

Добавим для полей **scEmail** и **scName** уникальные индексы (рис. 1.17).

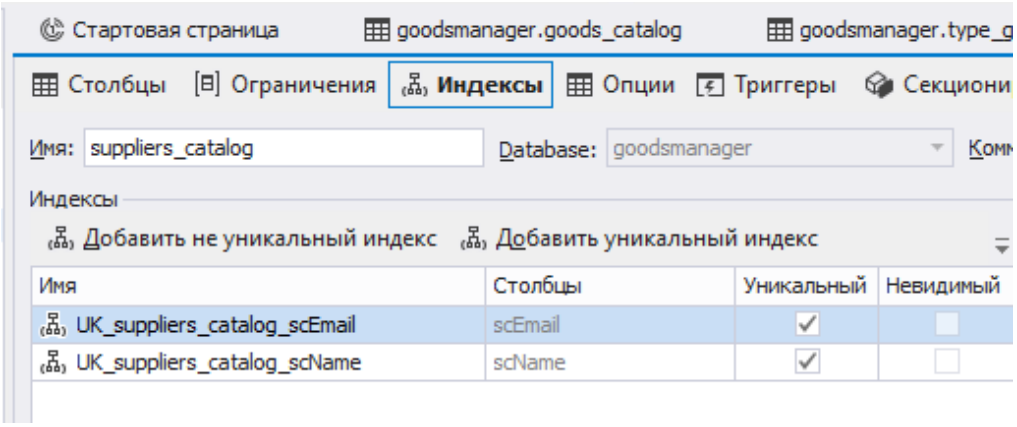



Рис. 1.17. Создание индексов для *suppliers_catalog*

Создадим две записи фирм-поставщиков в таблице **suppliers_catalog** (рис. 1.18).

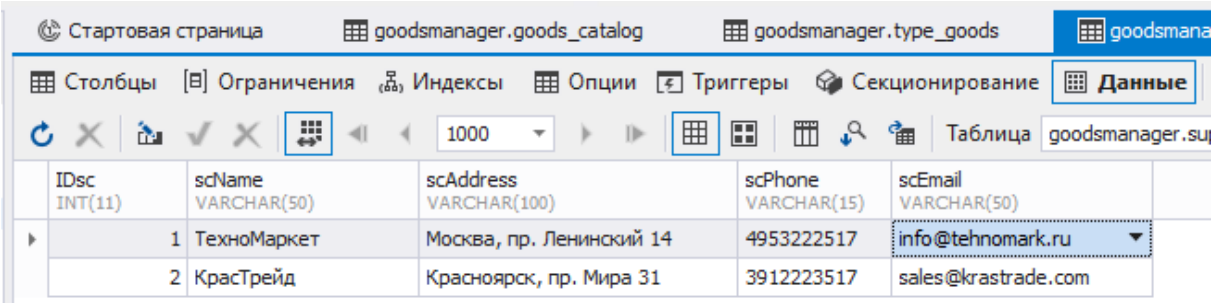



Рис. 1.18. Добавление данных в *suppliers_catalog*

Далее создадим таблицу **operations** отвечающую за операции с товарами (приход или продажа) (рис. 1.19), а также связь с **suppliers_catalog** (рис. 1.20).

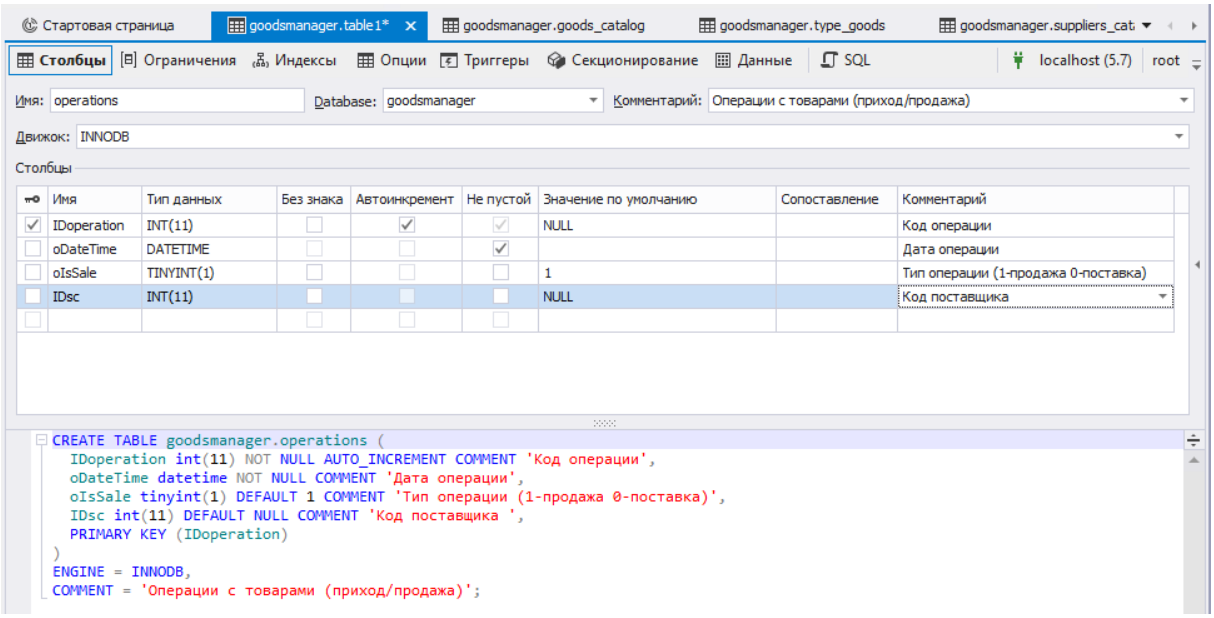



Рис. 1.19. Создание таблицы **operations**

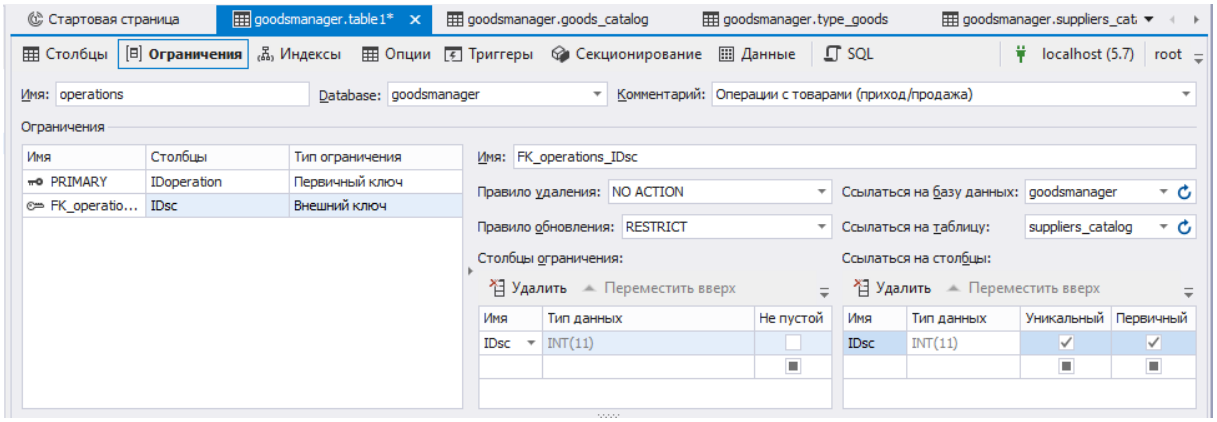



Рис. 1.20. Создание внешнего ключа для таблицы **operations**

Для ускорения поиска по дате добавьте неуникальный индекс для поля **oDateTime** (рис. 1.21).

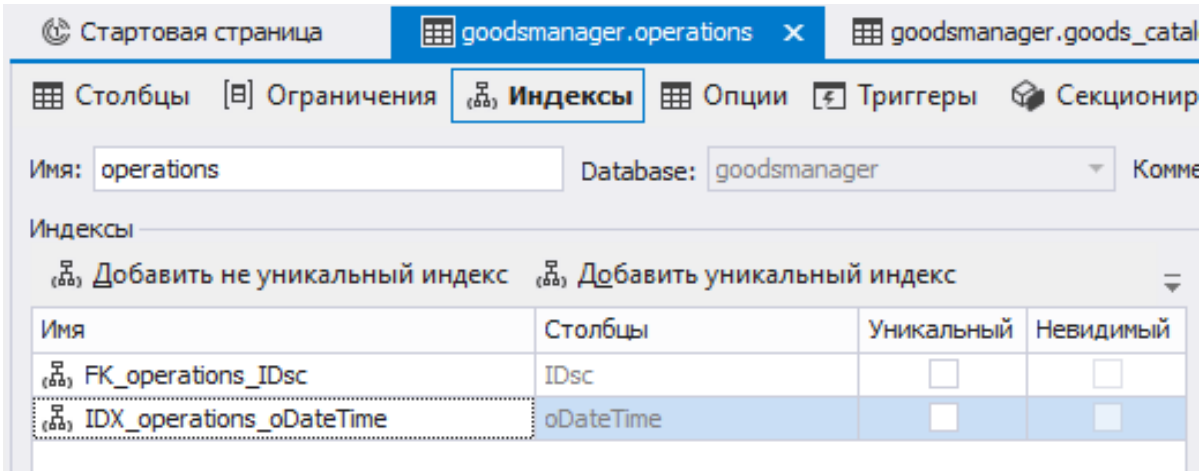



Рис. 1.21. Создание индекса для поля *oDateTime*

Далее необходимо создать вспомогательную таблицу **op_goods_list**, которая будет реализовывать связь многие-ко-многим между товарами и операциями, которые можно с ними совершить (рис. 1.22).

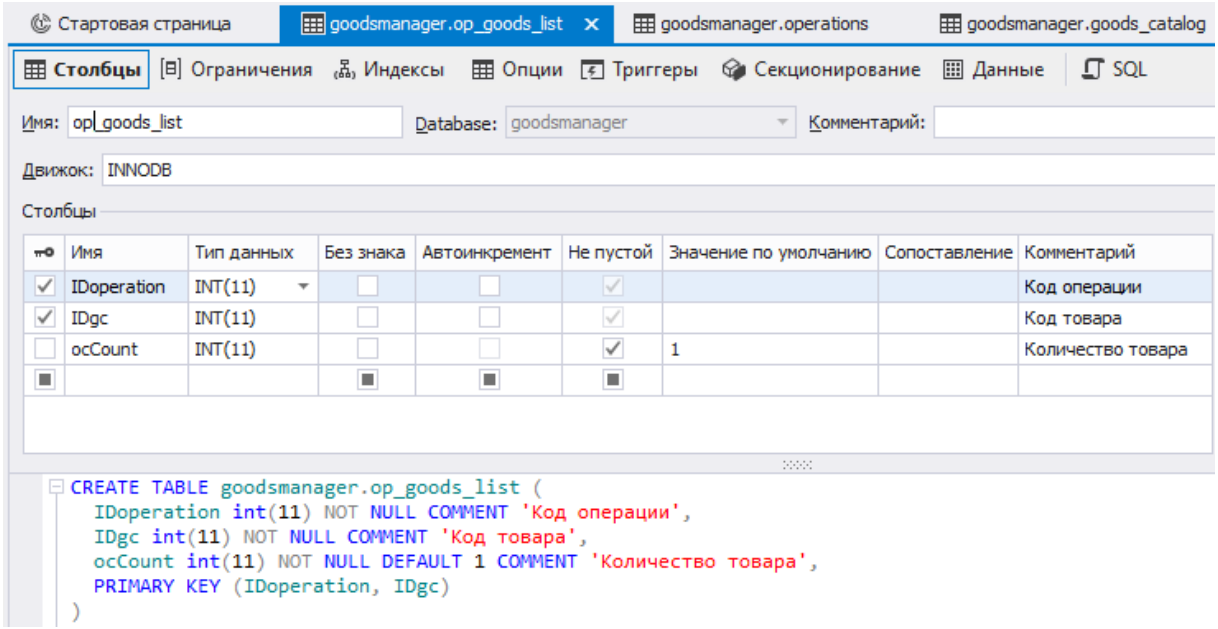
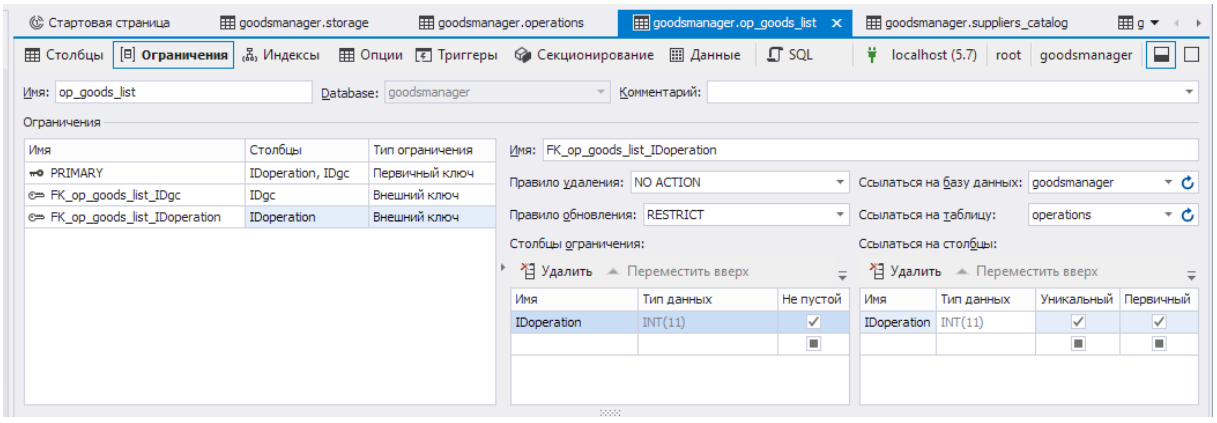



Рис. 1.22. Создание таблицы *op_goods_list*

Обратите внимание на составной первичный ключ состоящий из полей **IDoperations** и **IDgc**, более того эти поля будут использоваться для создания связей с таблицами **operations** и **goods_catalog** (рис. 1.23).



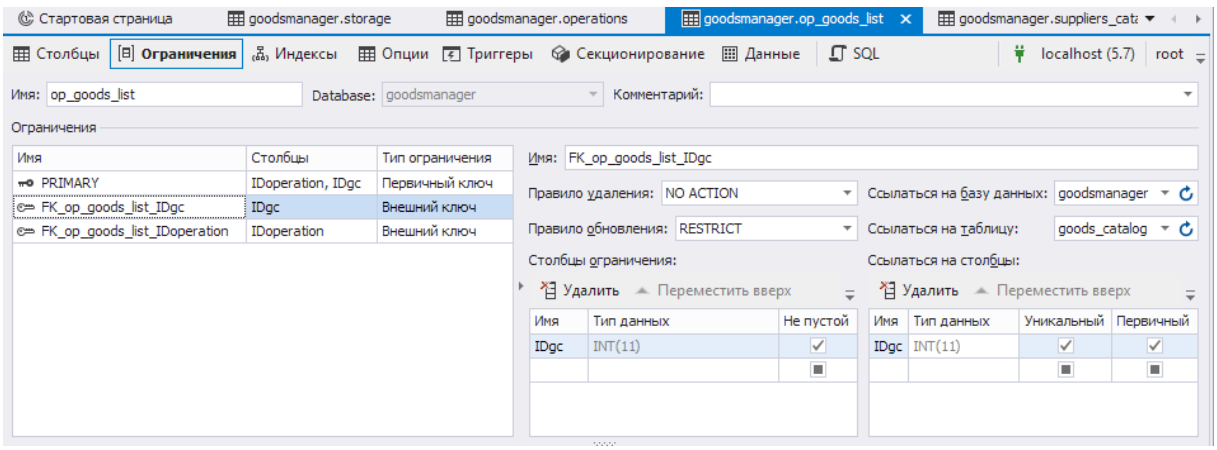



Рис. 1.23. Создание внешних ключей для таблицы *op_goods_list*

Заполним таблицу **operations** (рис. 1.24).

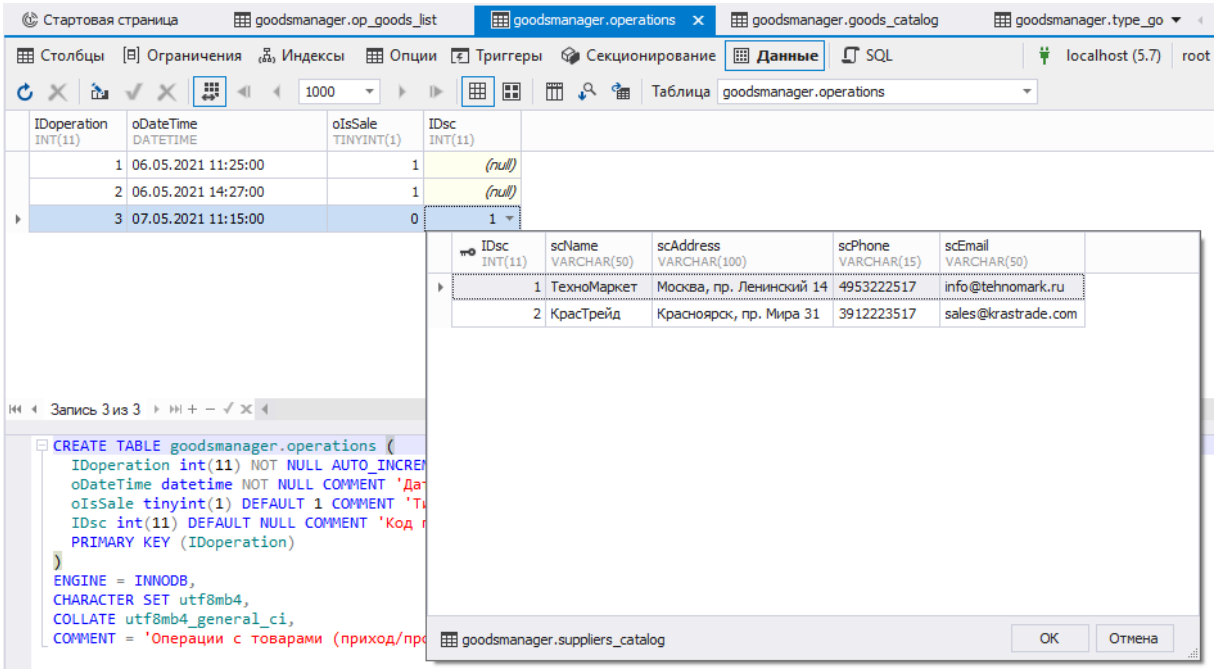
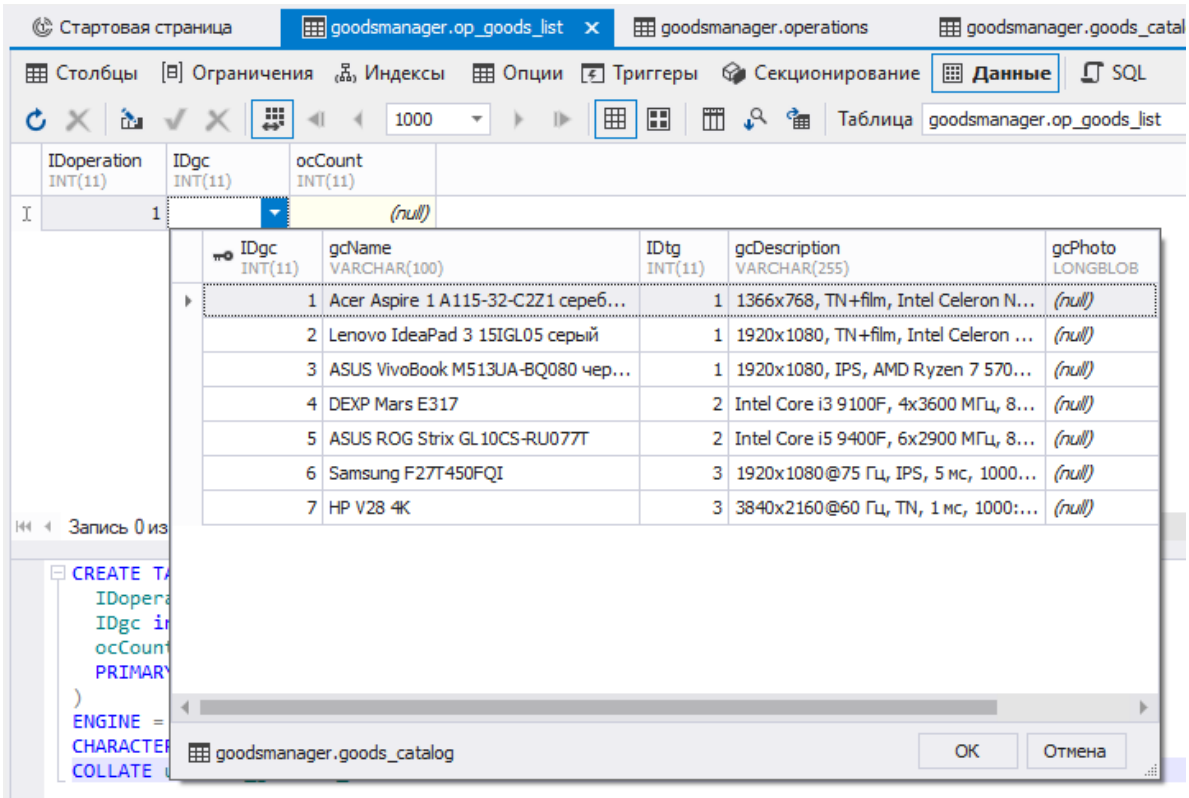



Рис. 1.24. Добавление записей в таблицу *operations*

Заполним таблицу **op_goods_list** (рис. 1.25).



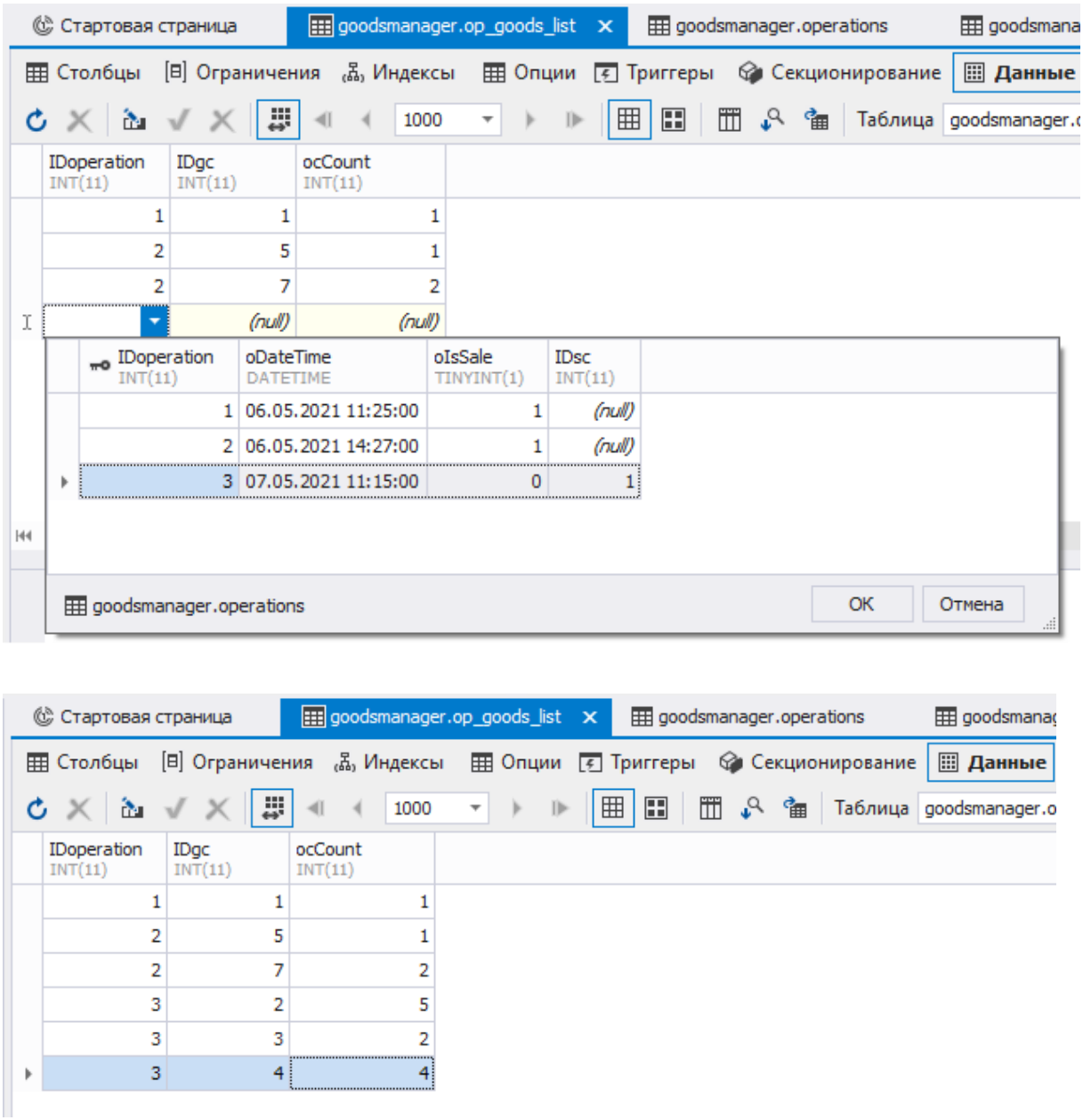






Рис. 1.25. Добавление записей в таблицу *op_goods_list*

Далее создадим дополнительную и последнюю таблицу **storage**, описывающую складские запасы магазина и необходимую в дальнейшем для демонстрации функциональных возможностей СУБД (рис. 1.26).

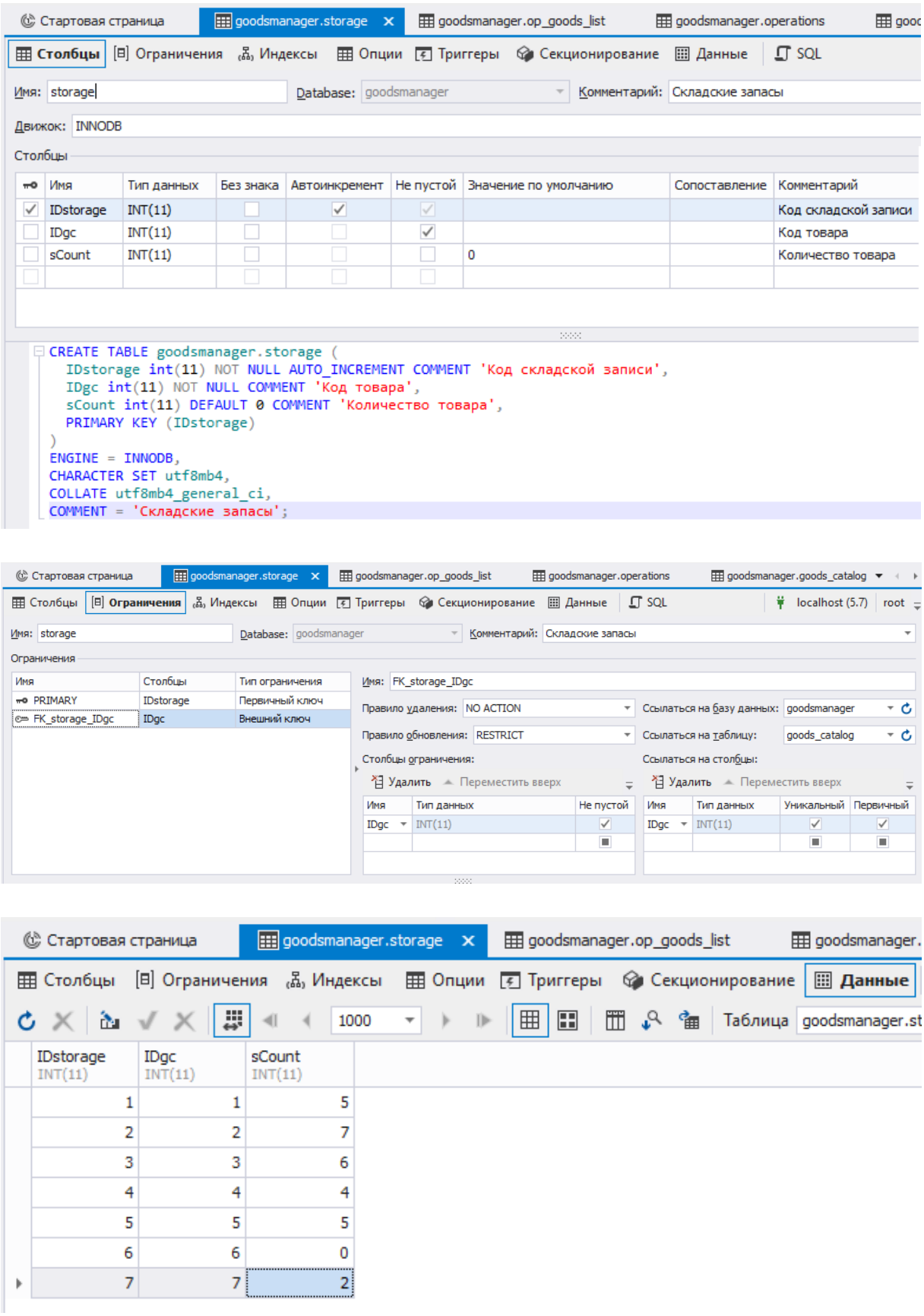



Рис. 1.26. Создание и заполнение таблицы *storage*

На этом этапе создание основных таблиц базы данных завершено. В дальнейшем в БД будут добавляться различные динамические элементы (представления, запросы, функции, триггеры и др.).

Теперь необходимо проверить корректно ли были созданы связи. Для этого в **dbForge Studio** выберите в основном меню пункт **База данных** и щелкните на **Диаграмма БД**. Перенесите одновременно все таблицы БД на открывшуюся область. Сгенерированная схема должна соответствовать физической диаграмме, представленной на рис. 1.27.

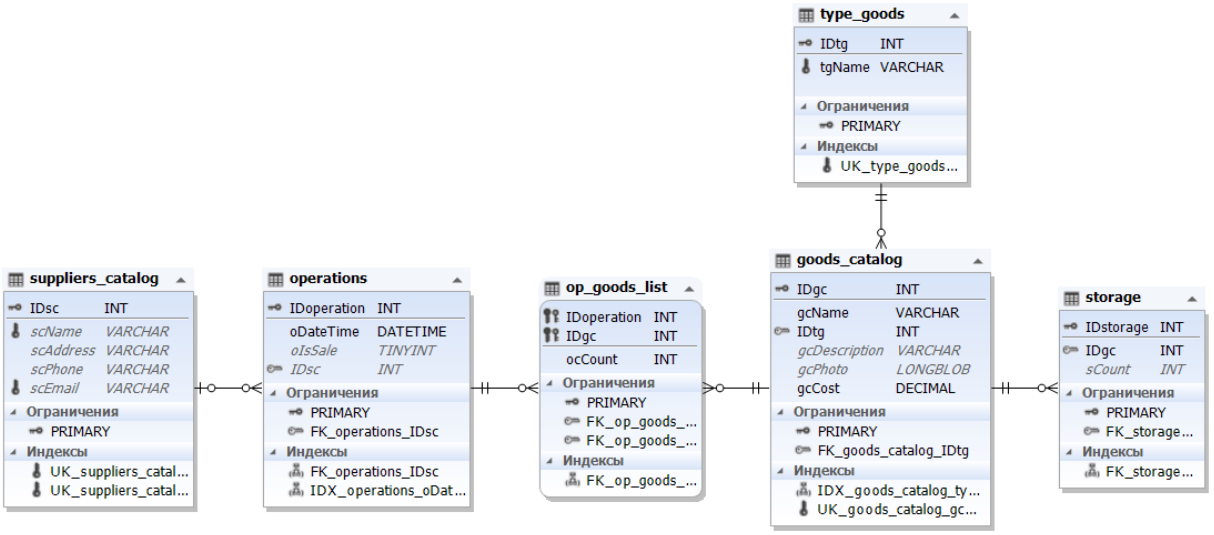



Рис. 1.27. Физическая схема базы данных

## 1.4. Создание представлений

**Представлением** (*view*) называется *SQL*-запрос на выборку, которому было присвоено имя и который затем сохранился в базе данных. Представление позволяет пользователю увидеть результаты сохраненного запроса, а язык SQL обеспечивает доступ к этим результатам таким образом, как если бы они были реальной таблицей базы данных.

Представления используются по следующим причинам:

– они позволяют сделать так, чтобы разные пользователи базы данных смогли видеть данные по-разному;

– с их помощью можно ограничить доступ к данным, разрешая пользователям видеть только некоторые из строк и столбцов таблицы;

– они упрощают доступ к базе данных, показывая каждому пользователю структуру хранимых данных в наиболее подходящем для него виде.

Создадим два представления, которые понадобятся в будущих лабораторных работах. Сгенерируем при помощи визуального конструктора новый SQL-запрос. Кликните на кнопку **Новый запрос** (рис. 1.28).

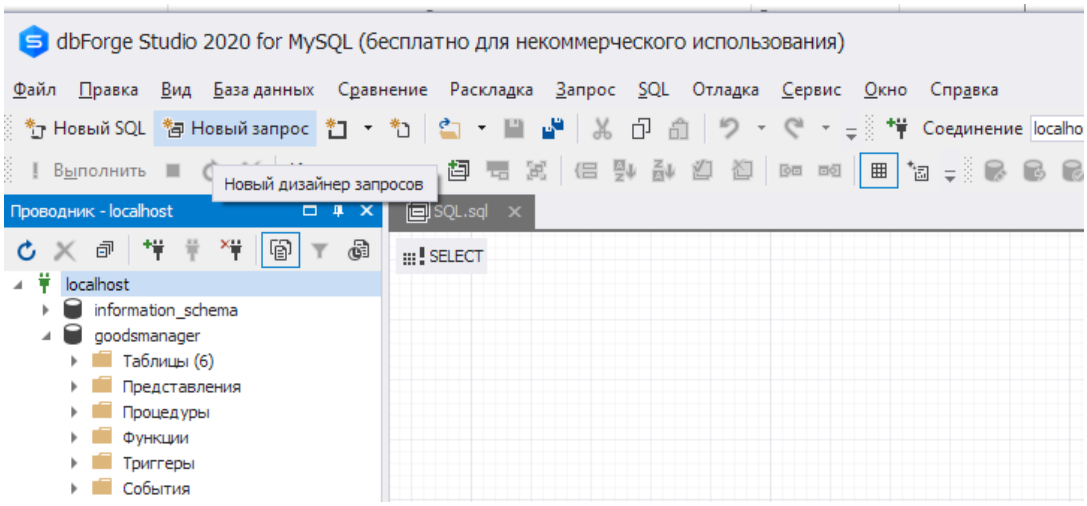



Рис. 1.28. Создание нового SQL-запроса

Перенесите на пустое поле таблицы: **goods_catalog**, **type_goods** и выберите необходимые поля, как показано на рис. 1.29.

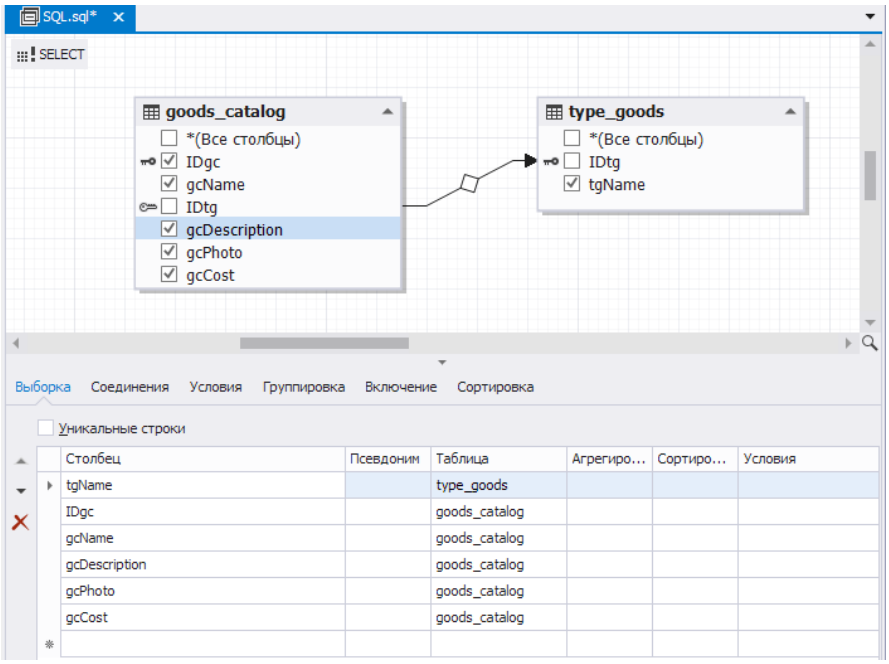



Рис. 1.29. Выбор необходимых полей таблицы

Нажмите на кнопку **Текст** и скопируйте получившийся SQL-запрос (рис. 1.30).

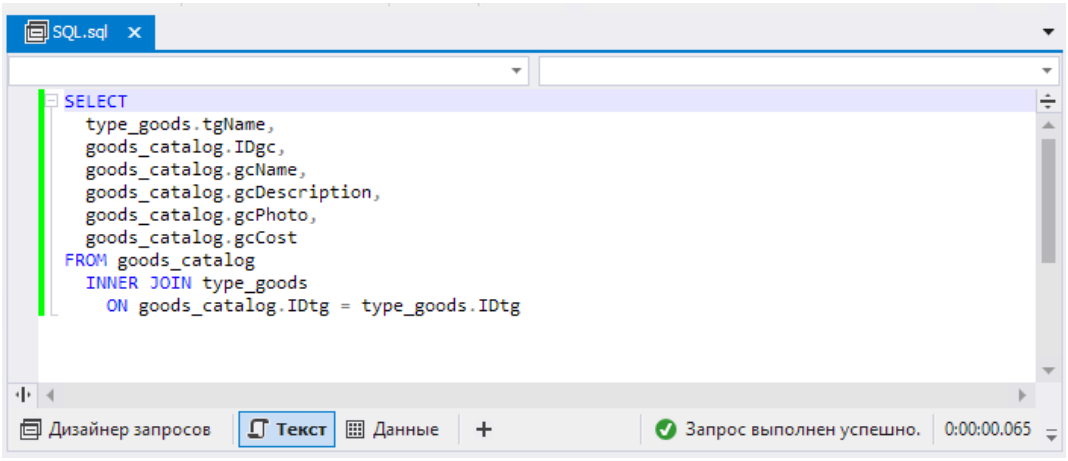



Рис. 1.30. Просмотр текста SQL-запроса

Создайте новое представление, дайте ему имя **view_goods_catalog**, вставьте в него сгенерированный SQL-запрос и нажмите **Применить изменения** (рис. 1.31).

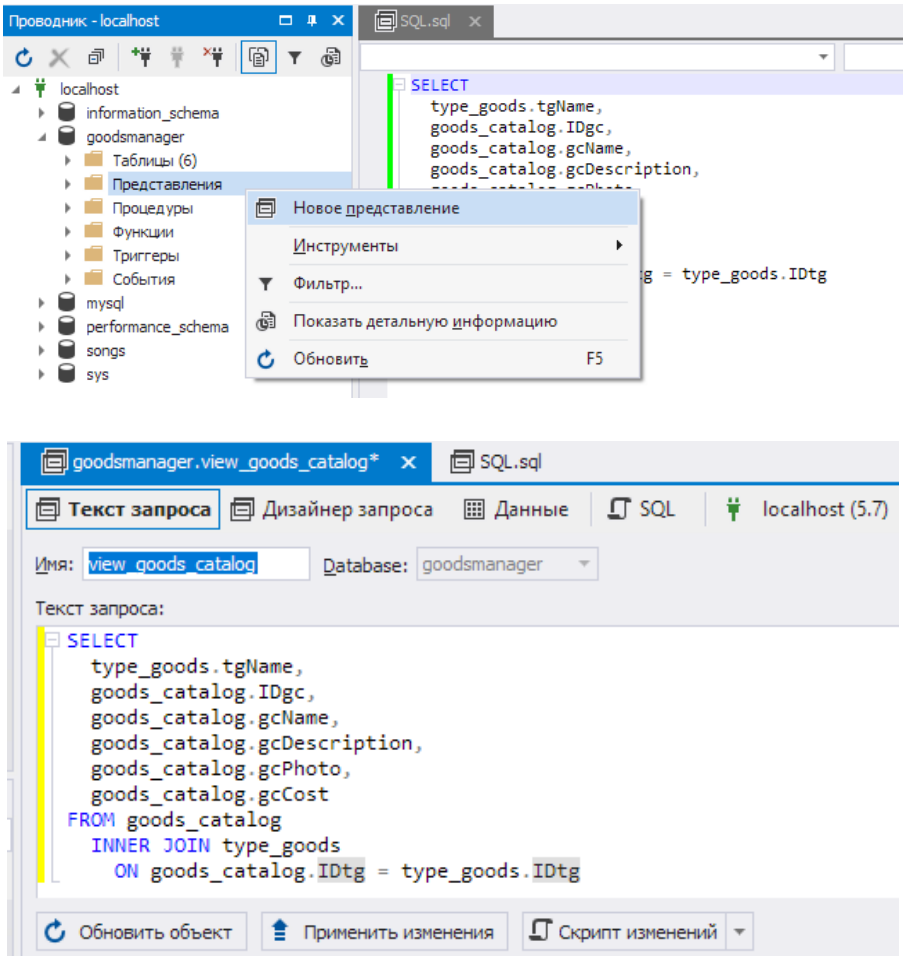
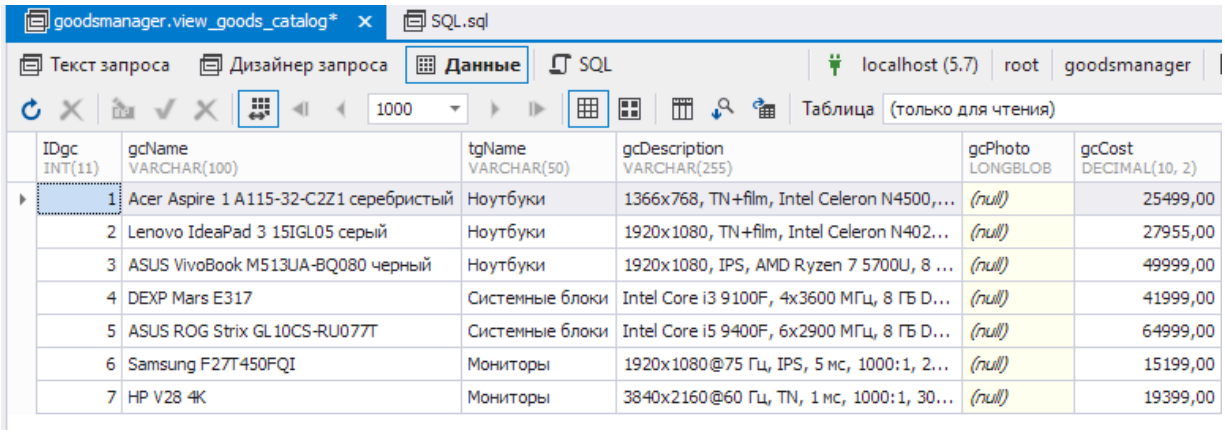



Рис. 1.31. Создание представления

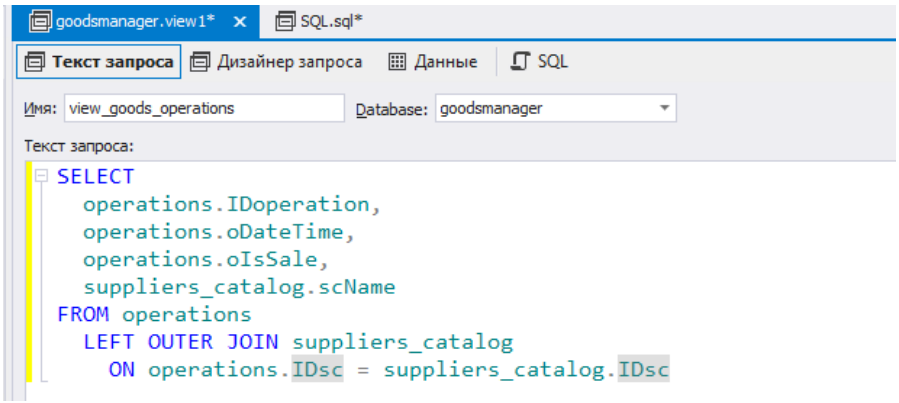
Теперь если кликнуть на кнопку **Данные** в представлении, можно увидеть результат выполнения SQL-запроса (рис. 1.32).

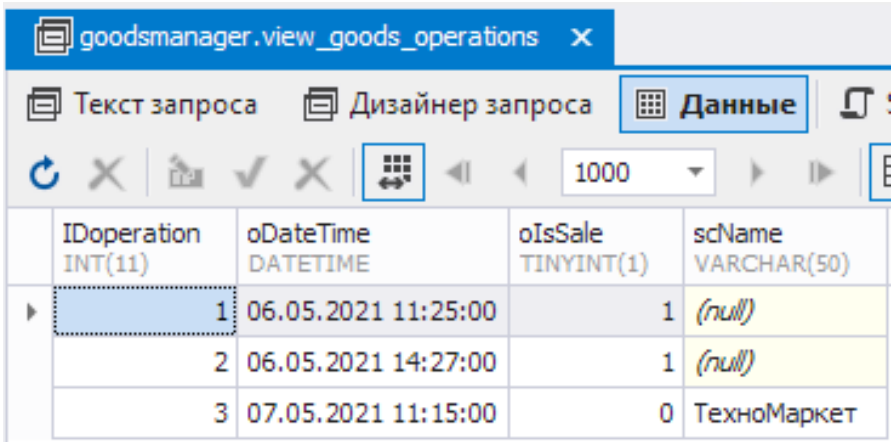



| IDgc INT(11) | gcName VARCHAR(100) | tgName VARCHAR(50) | gcDescription VARCHAR(255) | gcPhoto LONGBLOB | gcCost DECIMAL(10, 2) |
|---|---|---|---|---|---|
| 1 | Acer Aspire 1 A115-32-C2Z1 серебристый | Ноутбуки | 1366x768, TN+film, Intel Celeron N4500,... | (null) | 25499,00 |
| 2 | Lenovo IdeaPad 3 15IGL05 серый | Ноутбуки | 1920x1080, TN+film, Intel Celeron N402... | (null) | 27955,00 |
| 3 | ASUS VivoBook M513UA-BQ080 черный | Ноутбуки | 1920x1080, IPS, AMD Ryzen 7 5700U, 8 ... | (null) | 49999,00 |
| 4 | DEXP Mars E317 | Системные блоки | Intel Core i3 9100F, 4x3600 МГц, 8 ГБ D... | (null) | 41999,00 |
| 5 | ASUS ROG Strix GL10CS-RU077T | Системные блоки | Intel Core i5 9400F, 6x2900 МГц, 8 ГБ D... | (null) | 64999,00 |
| 6 | Samsung F27T450FQI | Мониторы | 1920x1080@75 Гц, IPS, 5 мс, 1000:1, 2... | (null) | 15199,00 |
| 7 | HP V28 4K | Мониторы | 3840x2160@60 Гц, TN, 1 мс, 1000:1, 30... | (null) | 19399,00 |

Рис. 1.32. Вывод данных представления

Далее создайте еще одно представление с именем **view_goods_operations** на основе SQL-запроса, показанного на рис. 1.33.




| IDoperation INT(11) | oDateTime DATETIME | oIsSale TINYINT(1) | scName VARCHAR(50) |
|---|---|---|---|
| 1 | 06.05.2021 11:25:00 | 1 | (null) |
| 2 | 06.05.2021 14:27:00 | 1 | (null) |
| 3 | 07.05.2021 11:15:00 | 0 | ТехноМаркет |

Рис. 1.33. Вывод данных представления *view_goods_operations*

На данном этапе создание базы данных магазина компьютерной техники завершено. Теперь необходимо создать резервную копию БД (рис. 1.34), которая позволит не только восстановить данные в случае сбоя, но и перенести их в другую реляционную СУБД помимо *MySQL*.

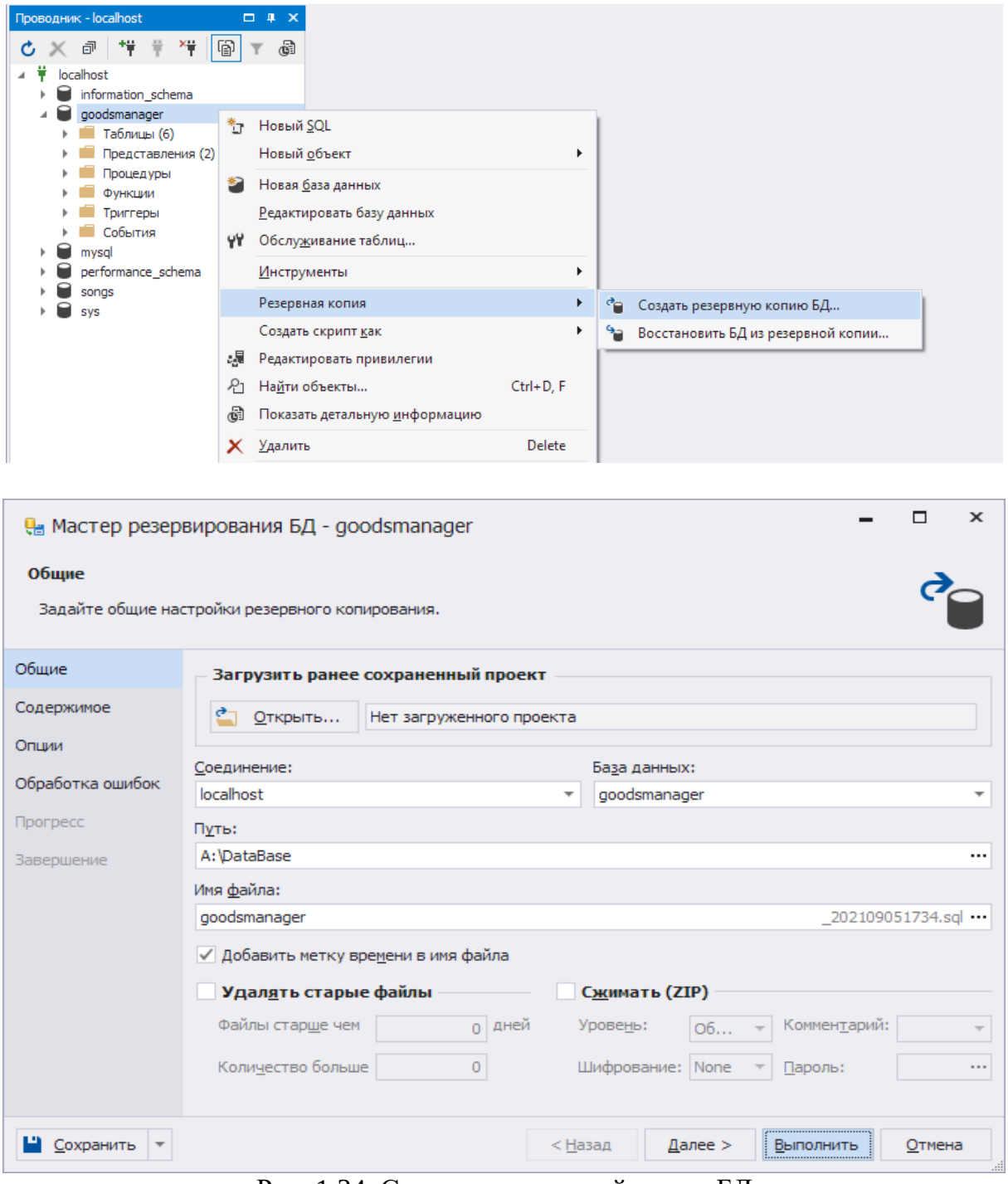



Рис. 1.34. Создание резервной копии БД

Резервная копия представляет собой обычный текстовый файл содержащий SQL-код и имеющий расширение **.sql**. Откройте файл с резервной копией и найдите в нем, как сохраняются данные из БД.

## КОНТРОЛЬНЫЕ ВОПРОСЫ

Что из себя представляет GPL лицензия?

Какая информация хранится в БД *mysql*?

Дайте характеристику *UTF*-8, чем она отличается от *utf8mb4*?

Чем *InnoDB* отличается от *MyISAM*?

Что такое *ACID*?

О чем говорит количество отображаемых значений для типа данных?

Охарактеризуйте типы данных с фиксированной точкой.

Что такое *UTC* и *timestamp* (что будет с этим типом данных '2038-01-09 03:14:07)?

Опишите тип данных *BLOB*, для чего он нужен?

Для чего создаются уникальные индексы. Для чего предназначены индексы?

Что из себя представляют внешние ключи?

Какие правила обновления/удаления связанных данных вы знаете?

Почему не стоит для правила обновления выставлять *CASCADE*?

Охарактеризуйте назначение механизма представлений.

Что из себя представляет резервная копия реляционной базы данных?