Лабораторная работа 7

РАСШИРЕННЫЕ ВОЗМОЖНОСТИ ПО РАБОТЕ С БАЗОЙ ДАННЫХ НА ОСНОВЕ SQL

Цель работы: ознакомиться с аспектами реализации комплексных связей в базе данных; изучить возможности применения хранимых функций и процедур, а также использования SQL для генерации пользовательских оповещений об ошибках; ознакомиться с возможностями встроенного профилировщика запросов; разработать приложение, для вывода результатов исполнения хранимых процедур.

ПОРЯДОК ВЫПОЛНЕНИЯ РАБОТЫ

1. Ознакомиться с теоретическими сведениями.
2. Создать в базе данных набор таблиц для хранения сведений о перемещении товаров между складами, а также изменить таблицы **operations** и **storage** для учета информации о складах.
3. Создать триггеры по работе со списком складов и выполнить модификацию ранее созданных триггеров.
4. Реализовать механизм для проверки возможности осуществления перемещения товара с одного склада на другой, с возможностью генерации сообщений об ошибках.
5. Реализовать представление для отображения сведений р перемещении товаров.
6. Разработать хранимую процедуру для выборки данных о полученных товарах.
7. Реализовать приложение для вызова процедур.
8. Просмотреть результаты профилирования запроса на перемещение товара.
9. Ответить на контрольные вопросы.

ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

При проектировании баз данных могут возникать ситуации, когда требуется организовать двойную связь между таблицами. Так наличие двойной связи необходимо в случае хранении в базе данных информации о перемещении объектов с одного места на другое.

В рамка данного лабораторного практикума используется предметная область складского учета. В связи с эти демонстрация реализации двойной связи между таблицами может быть продемонстрирована путем введения учета перемещение товаров с одного склада на другой.

Для реализации такой возможности нам понадобится модифицировать таблицы **Операции с товарами** и **Склад товаров**, а также добавить новые – **Перечень складов**, **Информация о перемещении товара** и **Состав перемещения**. Новые таблицы будут учитывать необходимые сведения для организации хранения информации о перемещениях товаров и сведения о складах (название и адрес).

Фрагмент логической модели базы данных затрагивающие изменения (с отображением всех необходимых связей) приведен на рисунке 7.1

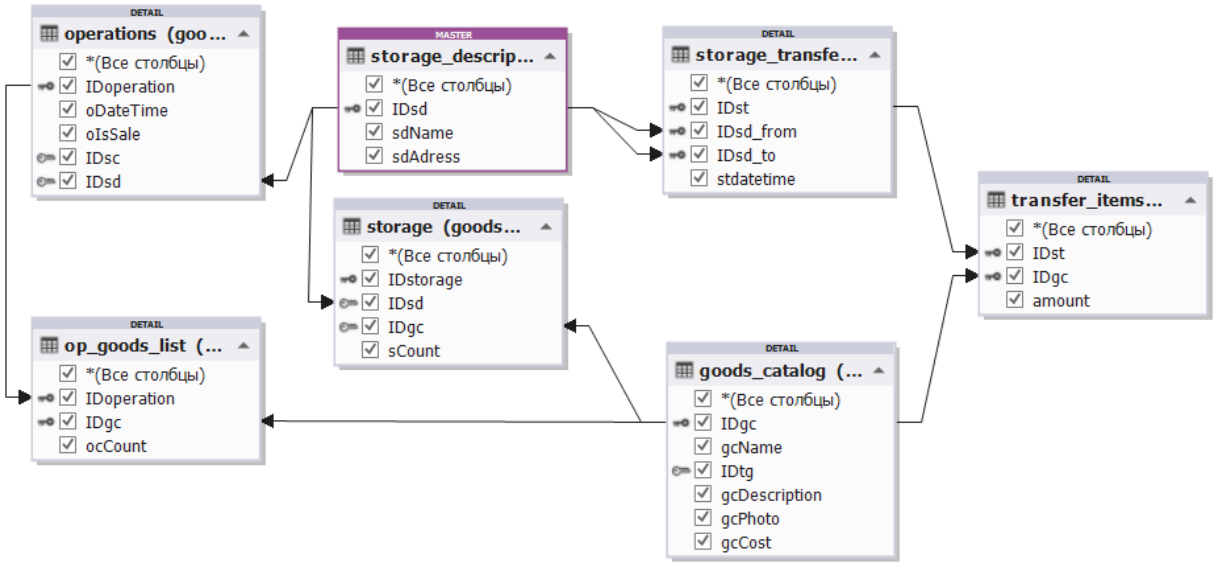


Рис. 7.1. Фрагмент логической модели базы данных

Рассмотрим, каким образом можно реализовать набор таблиц, для учета перемещения товаров, а также что нужно сделать для модификации функционирования бизнес-логики.

7.1 Модификация набора таблиц базы данных

В рамках рассматриваемого примера нам понадобится создать три новые таблицы: **Перечень складов**, **Информация о перемещении товара**, **Состав перемещения**. В таблице **Перечень складов** Структура вновь создаваемых таблиц приведена в табл. 7.1.

Таблица 7.1

Структура новых таблиц базы данных

Название поля	Тип поля	Дополнительные параметры	Ключ / индекс	Описание
storage_description (Перечень складов)				
IDsd	INT(11)	Автоинкремент	Ключ	Код склада
sdName	VARCHAR(80)	Не пустой, Уникальный	Индекс	Название склада
sdAdress	VARCHAR(255)	По умолчанию: NULL	–	Адрес склада

Окончание табл. 7.1

Название поля	Тип поля	Дополнительные параметры	Ключ / индекс	Описание
storage_transfers (Информация о перемещениях товара)				
IDst	INT(11)	Автоинкремент	Ключ ВК,	Код перемещения
IDsd_from	INT(11)	Не пустой	Ключ ВК, Индекс	Код склада, с которого осуществляется перемещение
IDsd_to	INT(11)	Не пустой	Ключ ВК, Индекс	Код склада, куда осуществляется перемещение
stDateTime	DATETIME	Не пустой	Индекс	Дата перемещения
transfer_items (Состав перемещения)				
IDst	INT(11)	Не пустой	Ключ ВК,	Код перемещения
IDgc	INT(11)	Не пустой	Ключ ВК, Индекс	Код товара
tiAmount	INT(11)	Не пустой По умолчанию: 1	–	Количество перемещаемого товара

Для начала создадим таблицу **storage_description** (Перечень складов). Структура таблицы в момент создания показана на рис. 7.2:

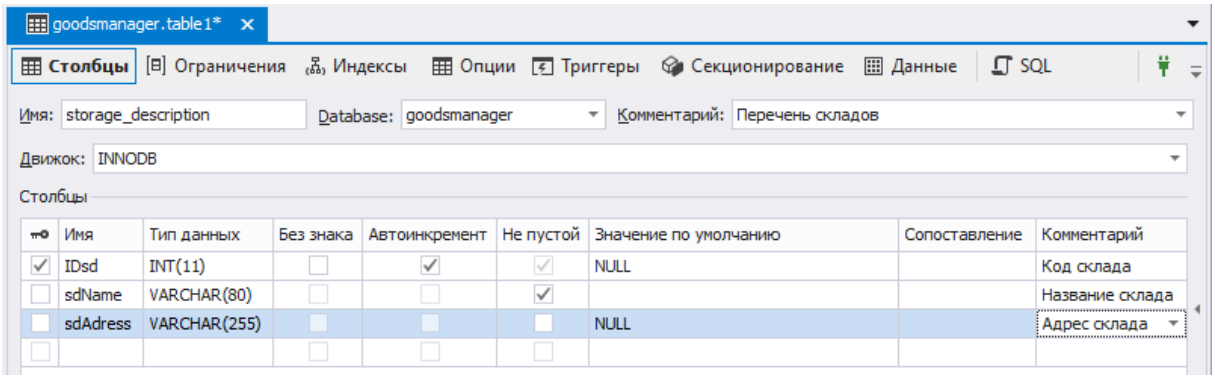


Рис. 7.2. Создание таблицы *storage_description*

Далее создадим таблицу **storage_transfers** отвечающую за сведения о перемещения товаров с одного склада на другой (рис. 7.3). Следует обратить внимание на наличие составного ключа по трем полям: **IDst**, **IDsd_from** и **IDsd_to**. Такой составной ключ позволит создать различные комбинации складов для перемещения и при этом будет уникальность хранения данных. В качестве альтернативы можно было бы создать первичный ключ по полю **IDst,** но в таком случае потребовалось бы создать

уникальный индекс для этих трех полей. В случае если уникальный индекс был бы создан только для полей **IDsd_from** и **IDsd_to**, то это могло бы привести к тому, что сведения об одинаковых перемещениях (склад отправления и склад получения товаров) в разные дни не смогли бы сохраниться в таблице.

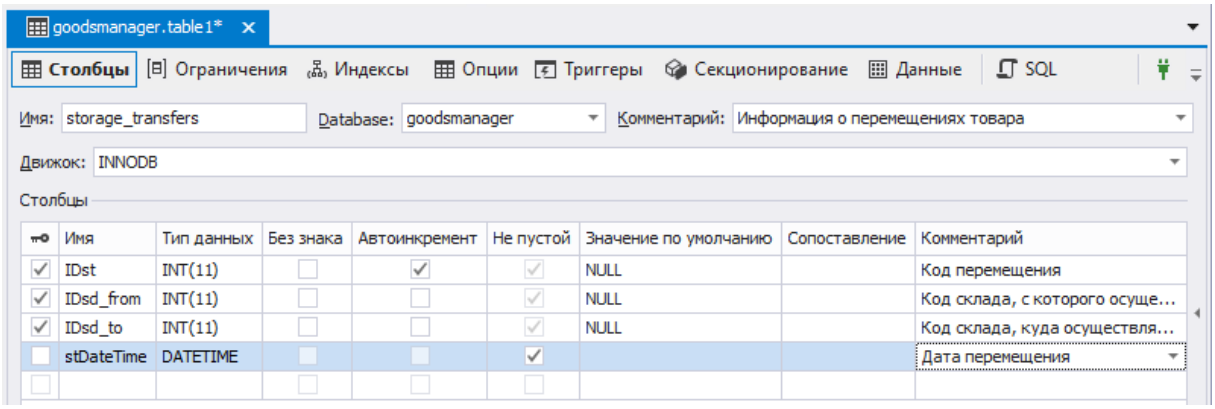


Рис. 7.3. Создание таблицы *storage_transfers*

Теперь создадим вспомогательную таблицу **transfer_items**, которая будет реализовывать связь многие-ко-многим между товарами и перемещениями товаров (Рис. 7.4). Данная таблица будет выполнять такую же роль что и таблица **op_goods_list**.

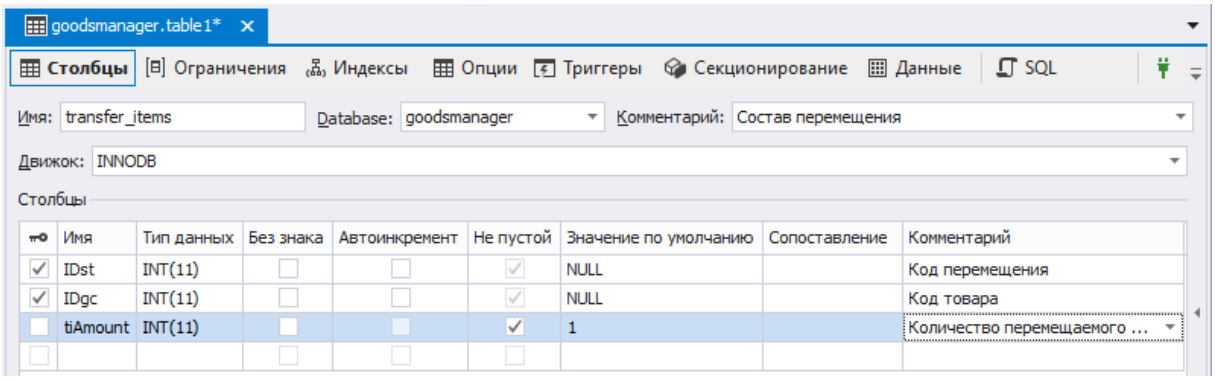


Рис. 7.4. Создание таблицы *transfer_items*

Следует обратить внимание на то, что здесь также используется составной первичный ключ по полям **IDoperations** и **IDgc**, при этом эти поля будут использоваться для создания связей с таблицами **storage_transfers** и **goods_catalog**. Настрои индексы в новых таблицах.

После создания новых таблиц можно перейти модификации существующих. Добавим в таблицы **storage** и **operations** поле **IDsd** и потребуем, чтобы поле не было пустым. Наличие такого ограничения желательно поскольку не даст создать запись без указания какого либо склада. Также для нового поля **IDsd** поставим значение по умолчанию **1** (Рис. 7.5). Установка значения по умолчанию необходима для того чтобы при создании

связей (ограничение в виде внешнего ключа) не возникла ошибка связанная с сопоставлением значений. После создания ограничений по внешнему ключу необходимо будет убрать значение по умолчанию.

При желании можно не накладывать подобные ограничения, но тогда контроль выбора склада будет возложен на сторону приложения. Без наличия требования не пустого значения и отсутствия значения по умолчанию позволит создать связь, но в таком случае придется в ручном режиме править данные, выбирая необходимый склад на основе созданной связи, для каждой записи в таблицах **storage** и **operations**.

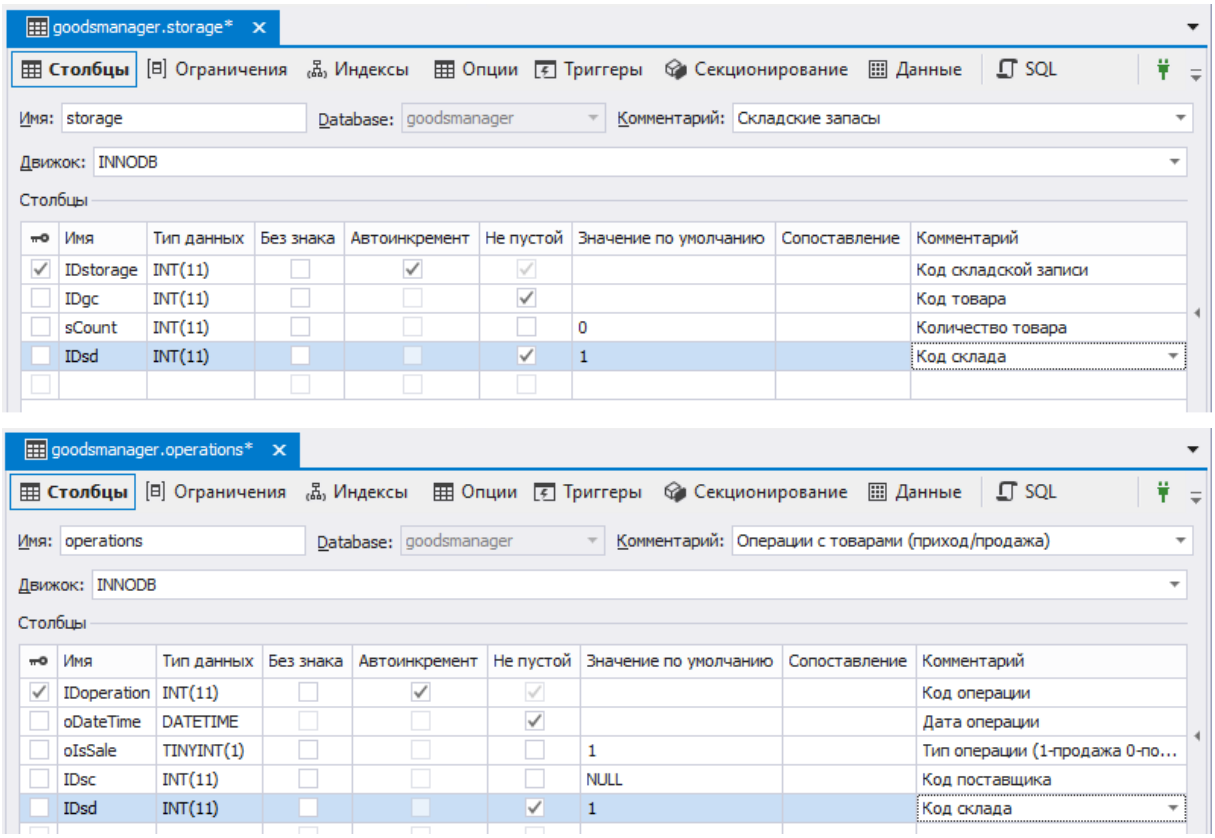
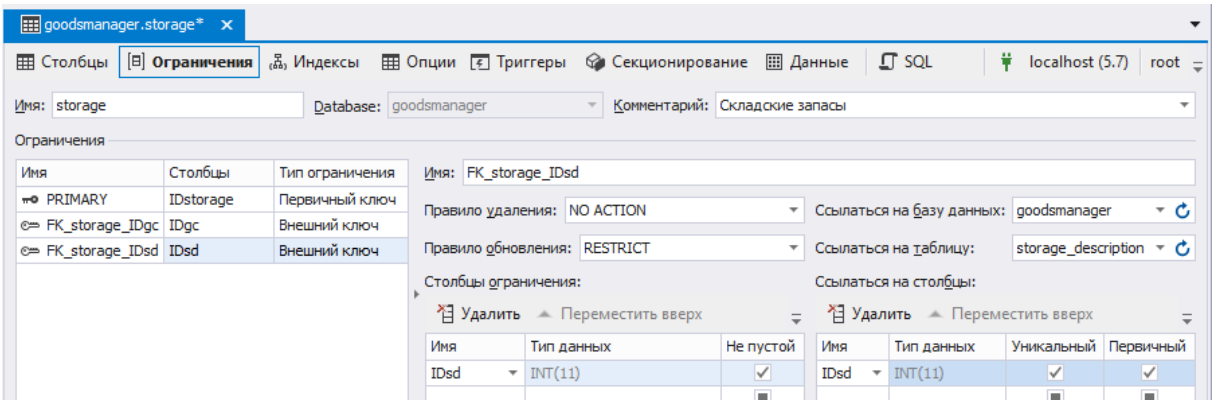


Рис. 7.5. Первичная модификация таблиц *storage* и *operations*.

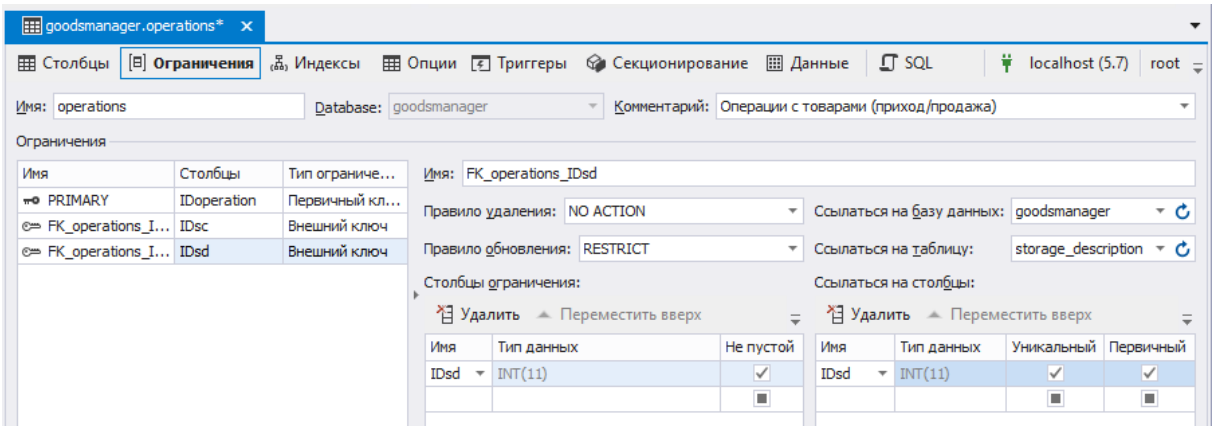
Теперь можно перейти к установке связей между таблицами базы данных. Перед настройкой связей необходимо добавить одну запись в таблицу **storage_description**, для того чтобы можно было нормально создать связи с существовавшими ранее данными в таблицах.

7.2 Установка ограничений для измененного набора таблиц

Сначала установим связь между таблицами **storage** и **storage_description**. Для этого в таблице **storage** создадим внешний ключ по полю **IDsd** (Рис. 7.6 а). После чего модифицируем ограничения в таблице **operations**, добавив внешний ключ по полю **IDsd** (Рис. 7.6 б).



а



б

Рис. 7.6. Модификация ограничений таблиц:
а – таблицы *storage,* б – таблицы *operations*.

Поскольку связи в таблицах **storage** и **operations** были успешно созданы, то необходимо в них для столбца **IDsd** изменить значение в колонке «значение по умолчанию» стерев имеющееся значение (**1**).

Теперь перейдем к настройке ограничений в таблице **storage_transfers** отвечающей за сведения о перемещениях товаров между складами. Создадим два внешних ключа которые будут связывать поля с **IDsd_from** и **IDsd_to** с полем **IDsd** из таблицы **storage_description**.

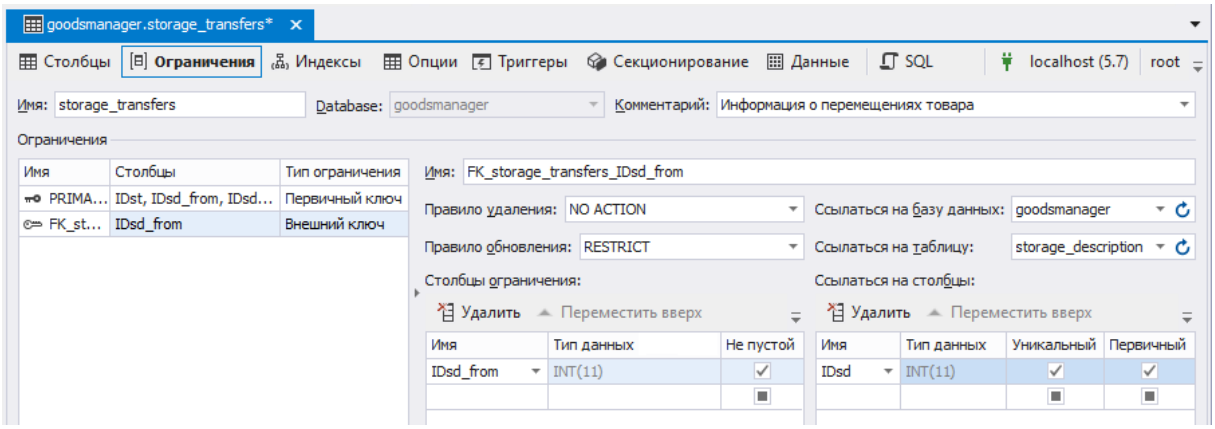


Рис. 7.6. Создание ограничения на поле **IDsd_from** в таблице *storage_transfers*.

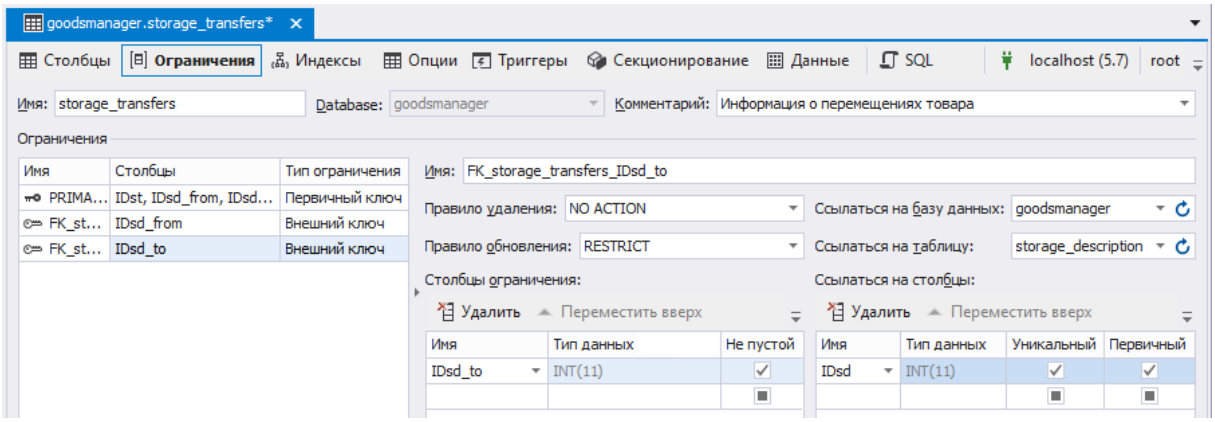


Рис. 7.7. Создание ограничения на поле **IDsd_to** в таблице *storage_transfers*.

Таким образом, два поля таблицы **storage_transfers** будут брать сведения из одной и той же таблицы **storage_description**.

Теперь осталось настроить ограничения в таблице **transfer_items,** которые нужно сделать по аналогии с тем, как это было сделано у таблицы **op_goods_list**. Результат настройки приведен на рисунке 7.8.

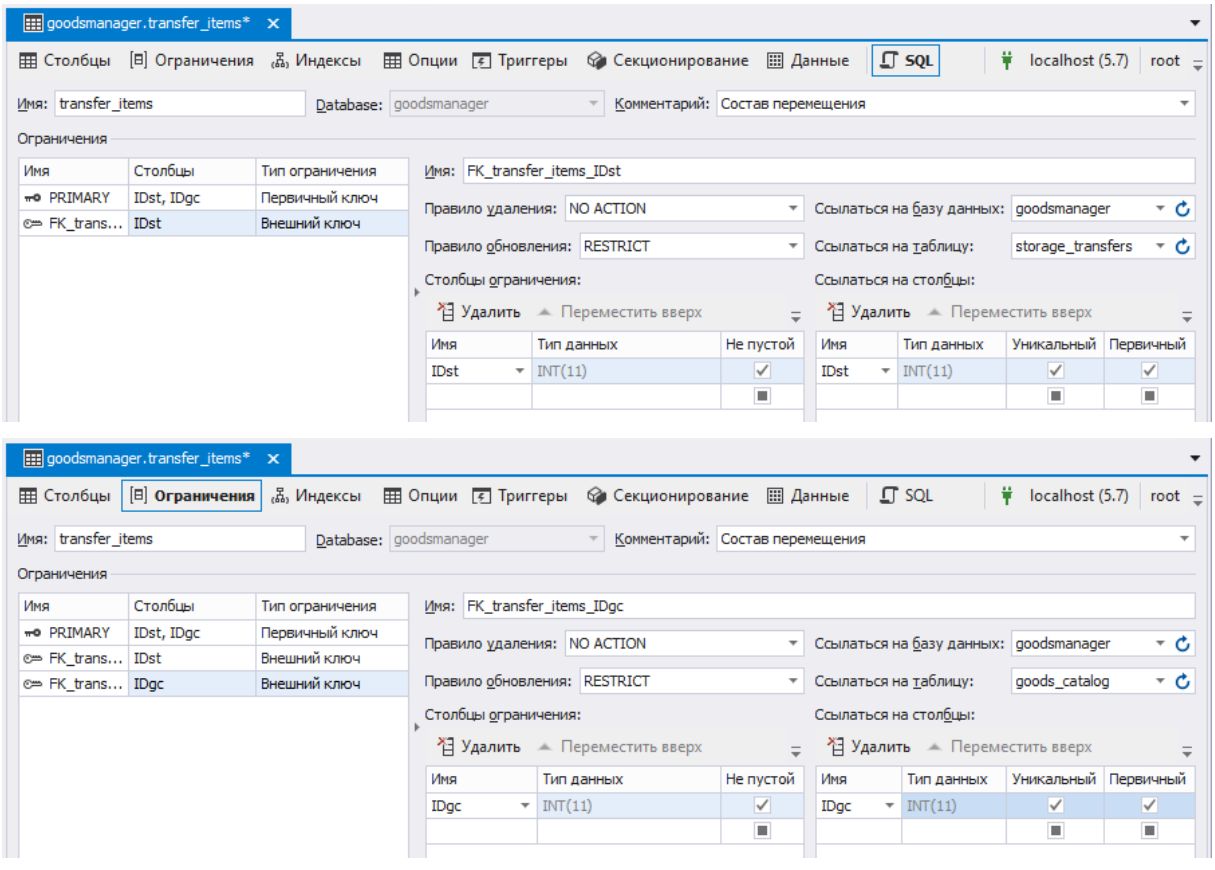


Рис. 7.7. Создание ограничений в таблице *transfer_items*.

При сохранении таблицы **transfer_items** появится сообщение о том, что для создания **внешнего ключа** нужно предварительно создать уникальный ключ в таблице **storage_transfers** (Рис. 7.8). Нажав на кнопку **Да**

вы подтвердите действие по автоматическому созданию уникального ключа для поля **IDst** в таблице **transfer_items**, что можно наблюдать на вкладке **Индексы** (Рис. 7.9).

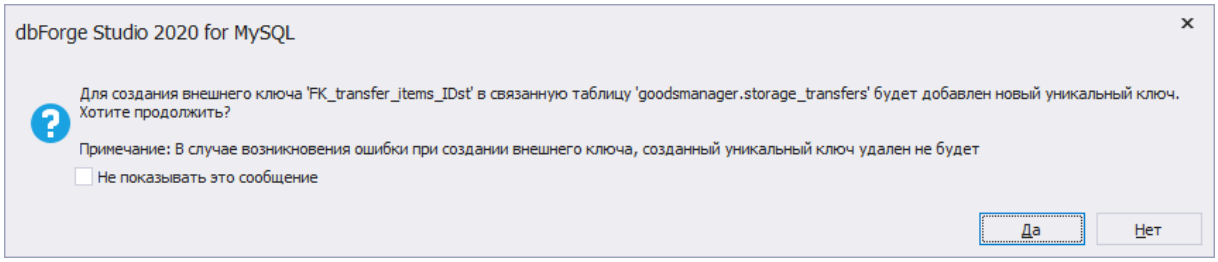


Рис. 7.8. Подтверждение добавления уникального ключа в таблице *storage_transfers*.

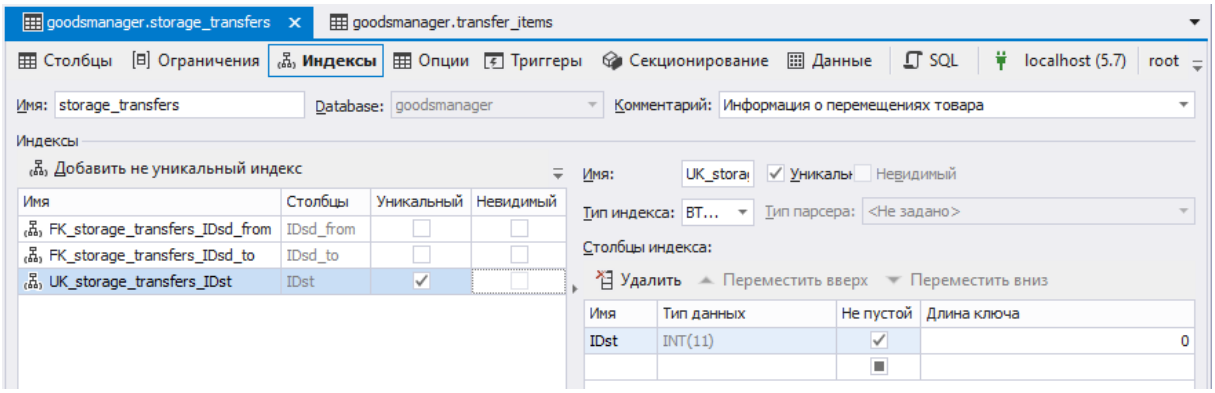


Рис. 7.9. Подтверждение добавления уникального ключа в таблице *storage_transfers*.

Все связи настроены и теперь при построении логической схемы через элемент **Связанные таблицы** пункта **Базы данных** основного меню мы сможем наблюдать связь как на рисунке 7.1. Далее перейдем к созданию и модификации триггеров реализующих бизнес-логику.

7.2 Модификация бизнес-логики

Сначала создадим триггер на добавление склада, который будет модифицировать таблицу **storage**, добавляя всю имеющуюся номенклатуру товаров для вновь создаваемого склада. Для этих целей будет производиться вставка записей на основе данных другой таблицы. Реализуется это путем применения в конструкции **INSERT** с оператором **SELECT**, при этом общий вид конструкции будет выглядеть следующим образом:

```
INSERT INTO Таблица_Назначения (поле1, поле2, поле3, ...)
SELECT поле1/константа1, поле2/константа2, поле3/константа3, ...
FROM Таблица_Источник
WHERE Условие_выбора; // Только если выбирается часть данных
```

Для нашего случая необходимо использовать три поля **IDsd**, **IDgc**, **sCount**, Для заполнения поля **IDsd** будет выступать значение триггера

NEW.IDsd, поле **IDgc б**удет заполняться из таблицы **goods_catalog**, а значение **sCount** всегда будет равно **0.** Создаваемый триггер **tr_add_new_storage** будет срабатывать после вставки записи в таблицу **storage_description,** и его настройка будет выглядеть, как показано на рисунке 7.10.

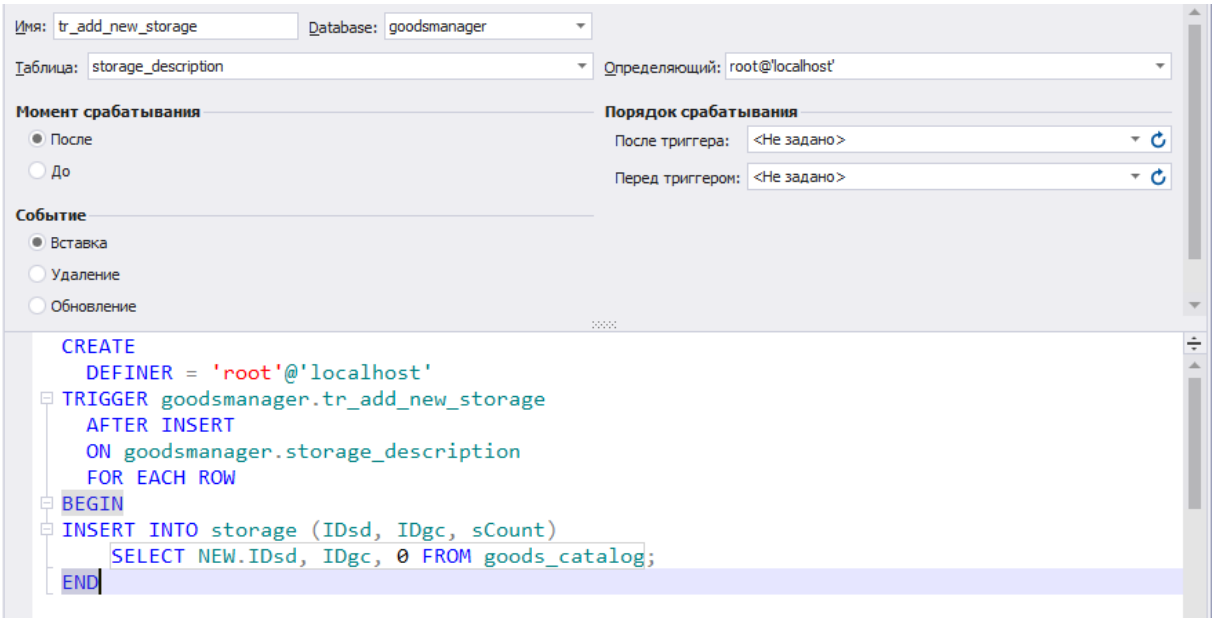


Рис. 7.10. Создание триггера *tr_add_new_storage*

Создадим триггер **tr_remove_existing_storage** для удаления складских записей если на складе не будет каких-либо товаров. Данный триггер будет срабатывать до удаления записи в таблице **storage_description**.

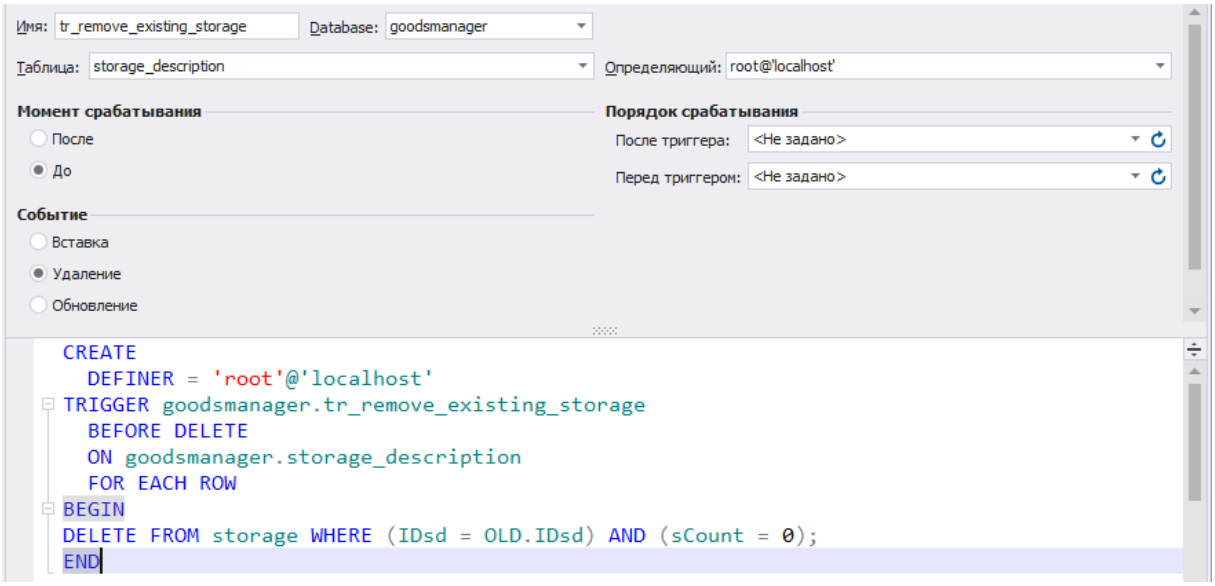


Рис. 7.10. Создание триггера *tr_remove_existing_storage*

Поскольку база данных стала учитывать различные складские помещения (разные склады), то нам придется модифицировать код триггера на добавление товара в каталог (**tr_add_goods_in_catalog**). Это необходимо, для того чтобы вновь добавленный товар учитывался на всех существующих складах в таблице **storage** со значением **0**. Реализация будет по аналогии с тем как было при создании триггера **tr_add_new_storage** (Рис. 7.11).

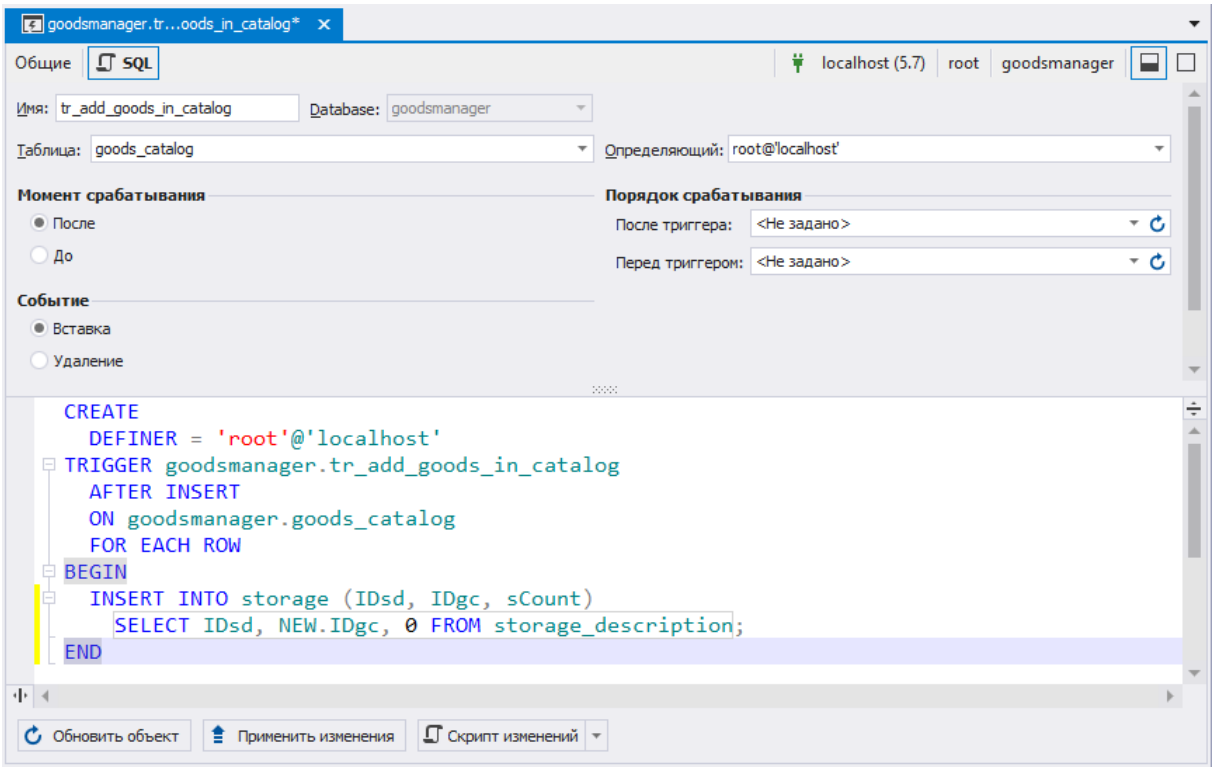


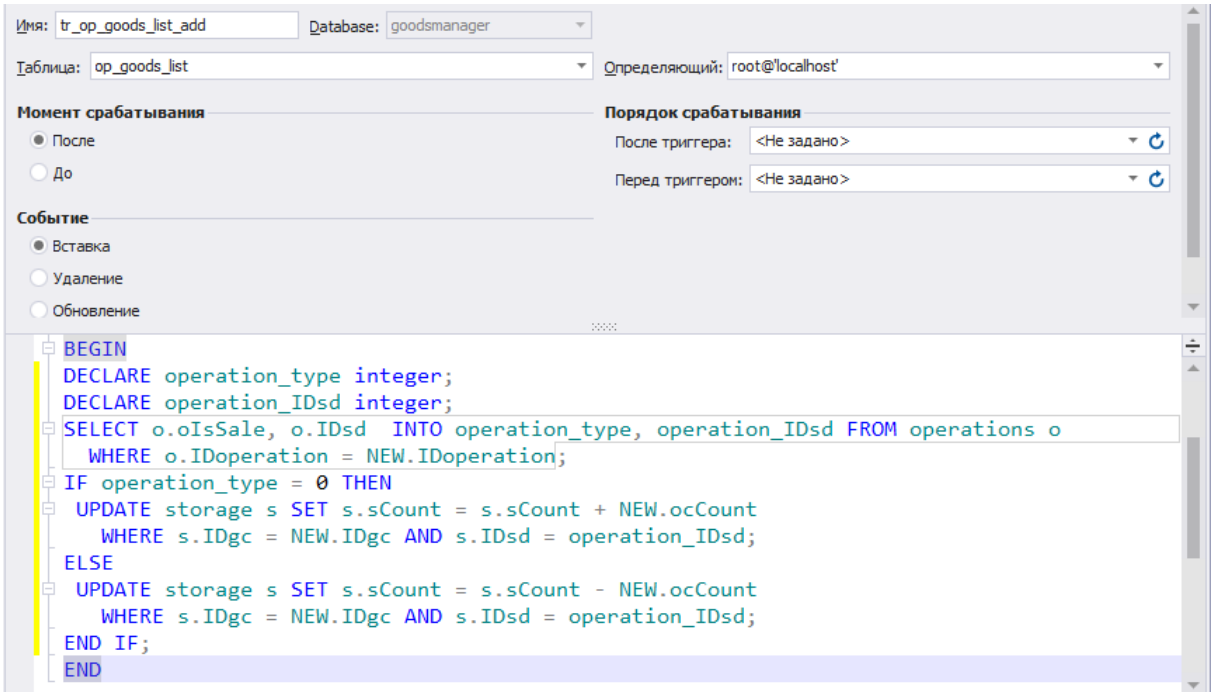
Рис. 7.11. Модификация триггера *tr_add_goods_in_catalog*

Триггер на удаление не придется изменять, поскольку он не даст удалить запись если хотя бы на одном складе есть товар с таким наименованием. В данном случае работает механизм транзакций и если на одном складе для удаляемого товара количество будет **0** а на другом складе несколько единиц, то будет предпринята попытка по триггеру удалятся записи однако запись с не нулевым количеством удаляемого товара будет обнаружена и в таком случае все действия в рамках транзакции откатятся.

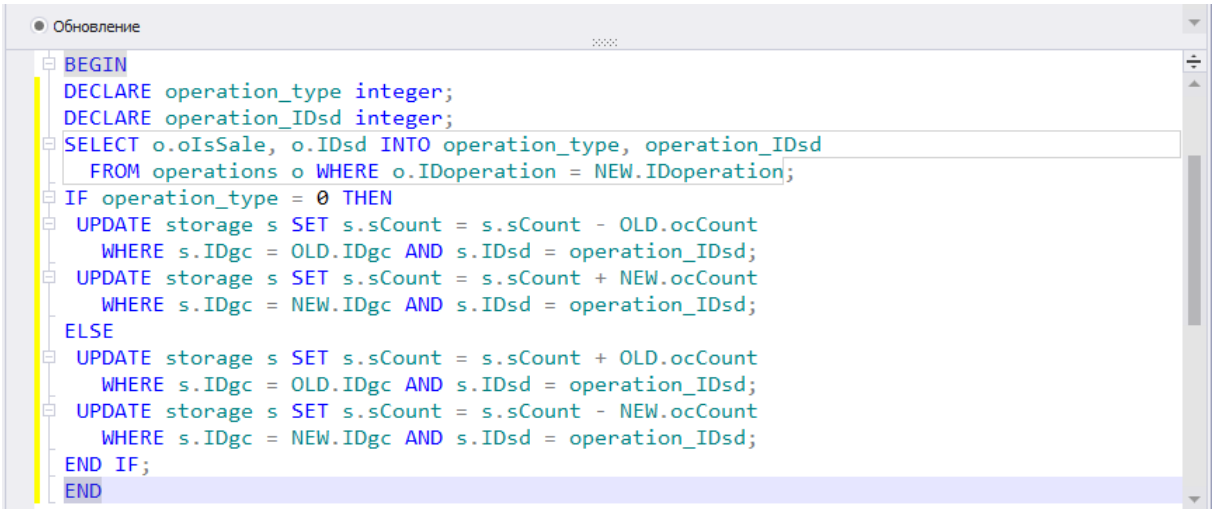
Поскольку было изменение в бизнес-логике по работе с товарами (учитываются разные складские помещения), то необходимо также выполнить модификацию триггеров по работе с составами операций. В рамках модификации добавляется новая переменная **operation_IDsd** отвечающая за идентификатор склада. Соответственно выборка данных также будет изменена. И примет следующий вид:

```
SELECT o.oIsSale, o.IDsd INTO operation_type, operation_IDsd
FROM operations o WHERE o.IDoperation = NEW.IDoperation;
```

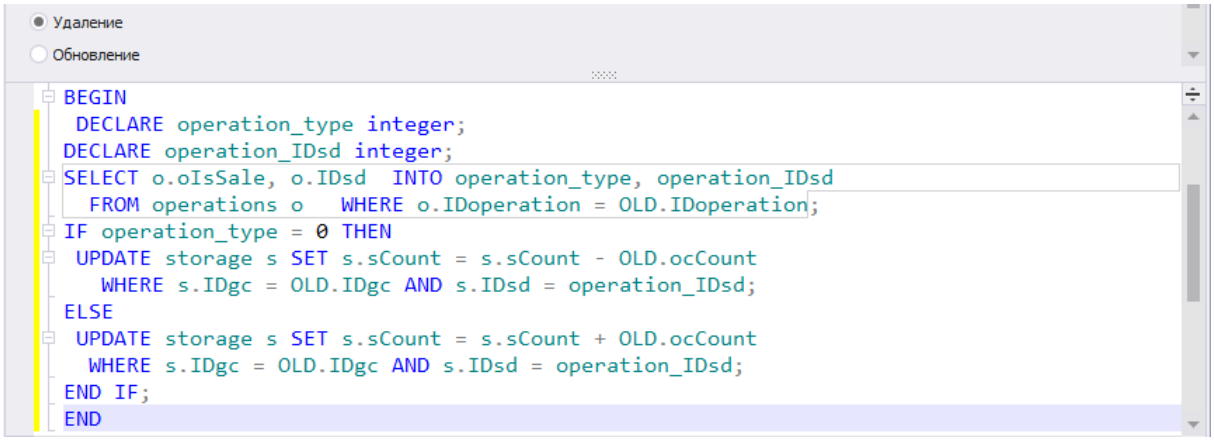
Пример измененного кода для триггеров показан на рисунке 7.12.



а



б



в

Рис. 7.12. Модификация триггеров: а – *tr_add_goods_in_catalog*; б – *tr_op_goods_list_edit*; в – *tr_op_goods_list_remove*.

На данный момент выполнены базовые модификации, которые позволят работать с операциями товаров. В реализованном функционале нет каких либо проверок для того чтобы операции выполнялись корректно. Далее будет рассмотрен вариант возможности работы бизнес-логики с механизмом проверки данных.

7.3 Бизнес-логика с контролем данных на уровне СУБД

Реализация механизма контроля вводимых данных будет рассмотрен на примере перемещения товаров между складами. Одной из возможных ошибок при добавлении записи моет быть указание одного склада в качестве склада отправки и получения товара. Для контроля значения также предполагается использование триггера (**tr_storage_transfer_validate)**, который будет проверять идентичны ли значения в полях **IDsd_from** и **IDsd_to** и в случае если значения будут совпадать, то будет генерироваться сообщение об ошибке.

В **MySQL** оператора **SIGNAL** является способом «вернуть» ошибку, он предоставляет информацию об ошибке обработчику, внешней части приложения или клиенту. Кроме того, оператор **SIGNAL** обеспечивает контроль над характеристиками ошибки (номер ошибки, значение **SQLSTATE**, сообщение). Общий вид синтаксической конструкции оператора может быть представлен по форме:

```
SIGNAL SQLSTATE Значение_SQLSTATE [дополнительная_информация]
```

Значение_SQLSTATE – задает строковую константу, представляющую значение SQLSTATE. Константа должна иметь длину 5 символов и должна удовлетворять правилам формирования значения SQLSTATE. Значение «45000» означает «Необработанное определяемое пользователем исключение» и рекомендуется для вывода пользовательских сообщений об ошибках.

Дополнительная информация представляет собой текст сообщения об ошибке и может быть задана в виде:

```
SET MESSAGE_TEXT = диагностическое_сообщение
```

Для выполнения оператора **SIGNAL** в **MySQL** не требуются какие-либо особые привилегии.

В нашем случае код триггера **tr_storage_transfer_validate** будет выглядеть следующим образом:

```
IF NEW.IDsd_from = NEW.IDsd_to THEN
SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = "Невозможно использование
одного и того же склада в качестве параметров перемещения";
END IF
```

Демонстрация настройки триггера приведена на рисунке 7.13.

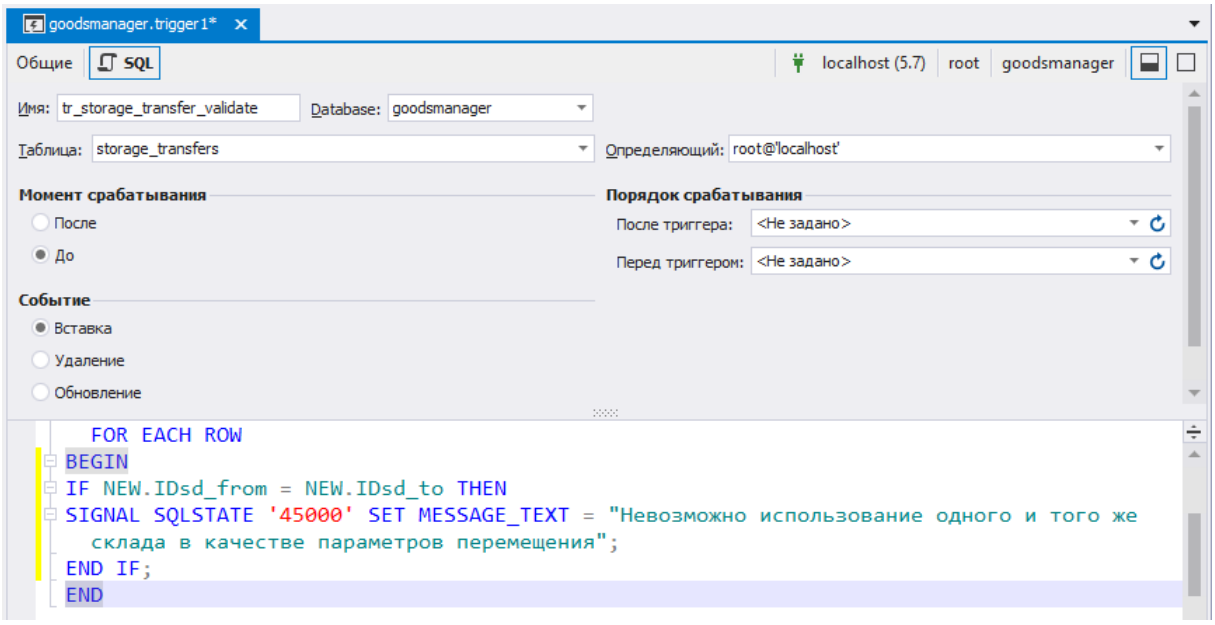


Рис. 7.13. Создание триггера *tr_storage_transfer_validate*

Для проверки работоспособности данного механизма добавим в базу данных сведения еще об одном складе и попробуем заполнить таблицу, установив полям **IDsd_from** и **IDsd_to** одинаковое значение. При попытке сохранения нам будет показано сообщение (Рис. 7.14).

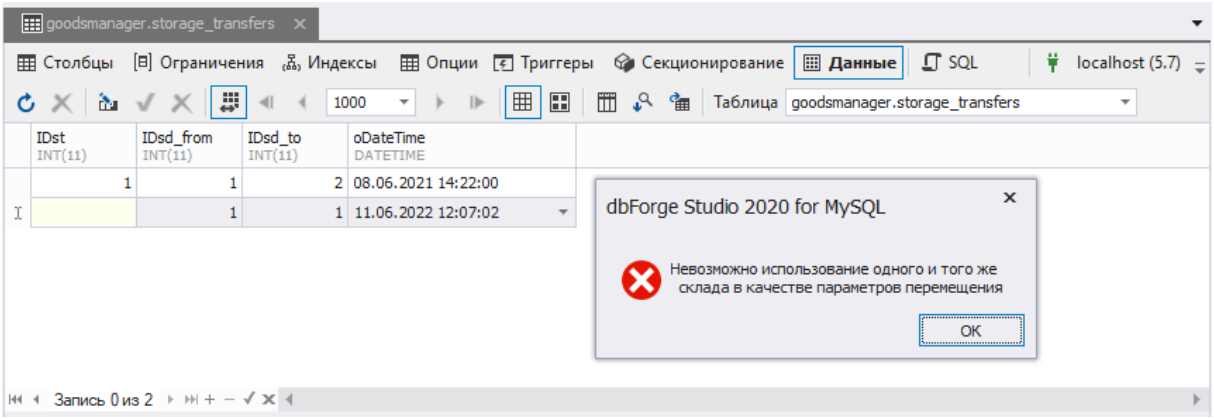


Рис. 7.14. Пример работы механизма проверки

С целью упрощения проверки доступности товара для перемещения целесообразно реализовать функцию **f_check_StorageAmount** у которой будут параметры типа **INT**:

– идентификатор склада (**p_IDsd**),
– идентификатор товара (**p_IDgc**)
– количество товара (**p_amount**).

В результате проверки функция будет возвращать значение **1,** когда на складе достаточное количество товара либо значение **0** (количество товара не достаточно). Настройка функции **f_check_StorageAmount** показана на рисунке 7.15.

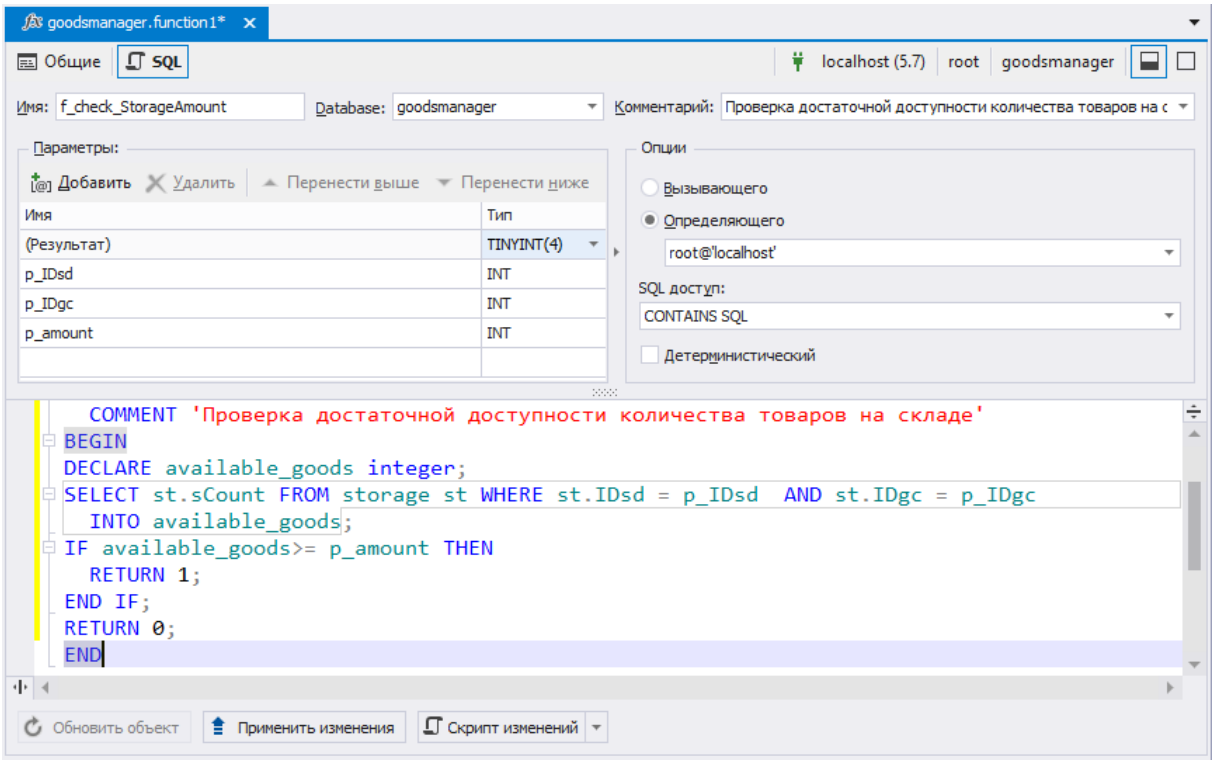


Рис. 7.15. Настройка функции *f_check_StorageAmount*

Эта функция будет использоваться для проверки доступности товара при выполнении операций добавления и удаления записей в таблице **transfer_items**. Проверку и действия будут выполнять соответствующие триггеры.

Создадим триггер **tr_transfer_items_add** который будет проверять доступность товара на складе, с которого будет осуществляться перемещение (Рис. 7.16). В коде этого триггера можно увидеть проверку

```
IF f_check_StorageAmount(IDsd_from, NEW.IDgc, NEW.amount) THEN
 BEGIN
  /*Изменение данных товаре на указанных складах*/
 END;
 ELSE
 /*Вывод пользовательского сообщения об ошибке*/
 SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = "Количество товара
недостаточно для перемещения";
 END IF;
END
```

Аналогичная проверка будет и при реализации триггера **tr_transfer_items_remove** (Рис. 7.17), отвечающего за проверку при удалении записи из таблицы состава перемещений **transfer_items**.

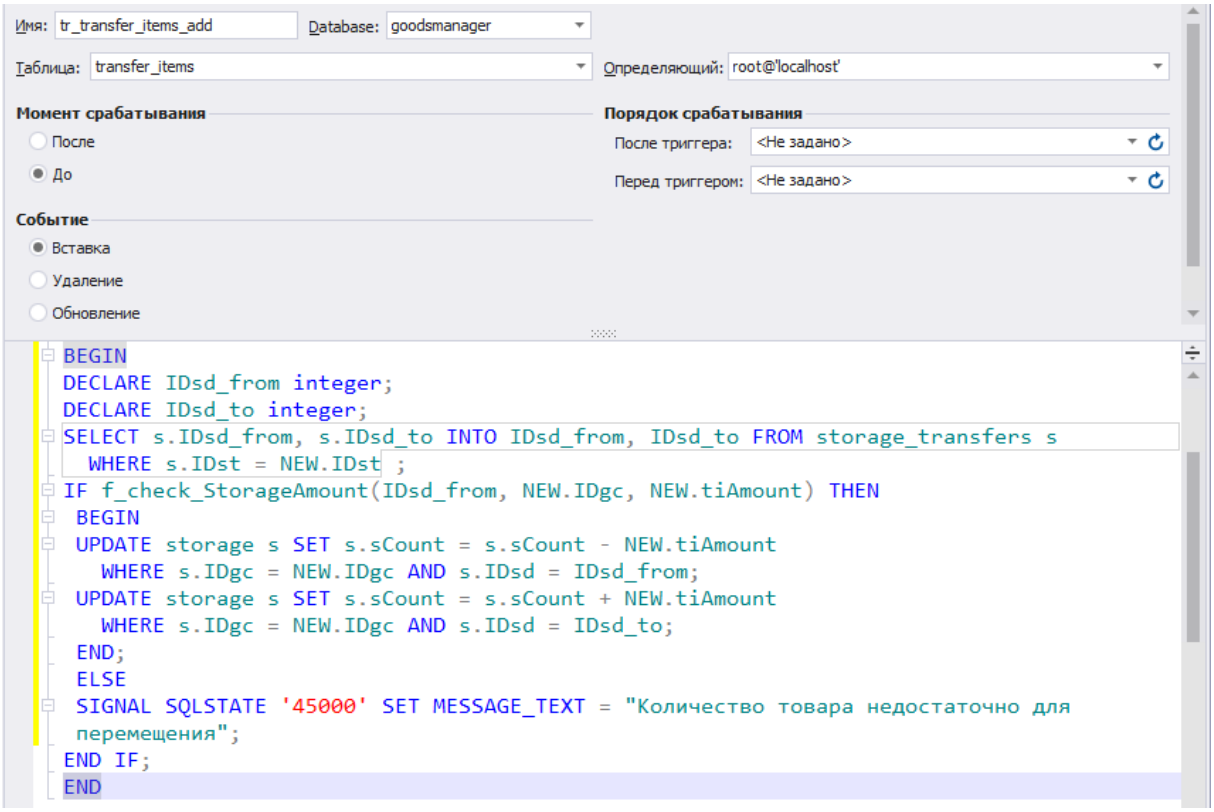


Рис. 7.16. Создание триггера *tr_transfer_items_add*

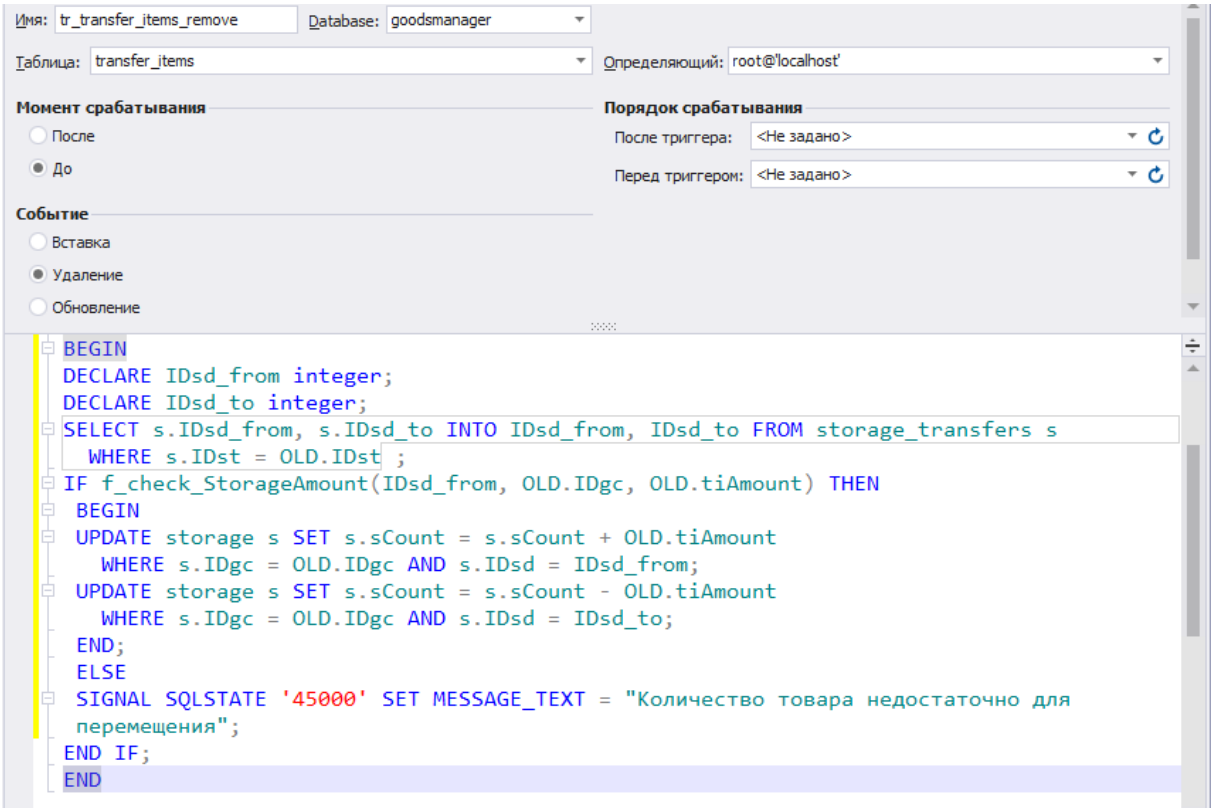


Рис. 7.17. Настройка функции *tr_transfer_items_remove*

При попытке совершить перемещение товара в количестве большем чем содержится на складе отправления будет выведено сообщение об ошибке (Рис. 7.18).

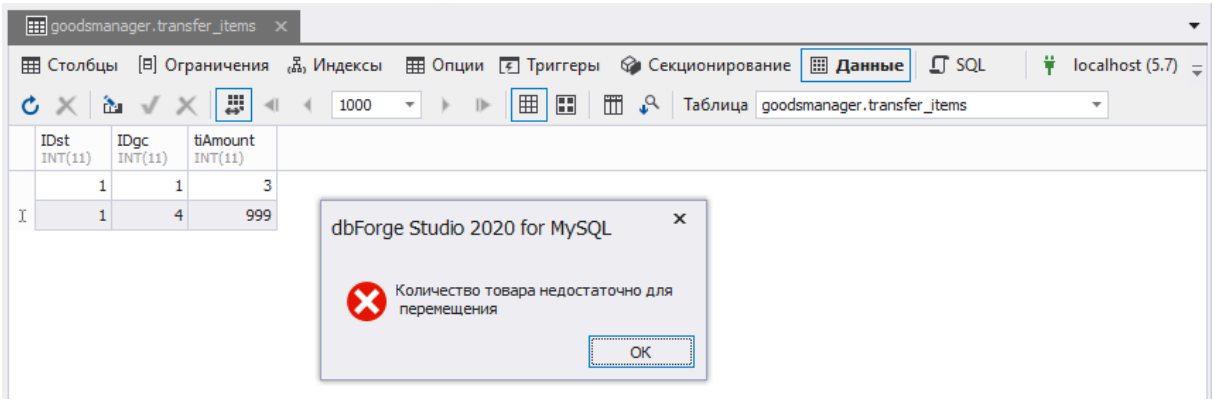


Рис. 7.18. Пример сообщения об ошибке

По аналогии можно сделать контроль значений при изменении количества товара. При этом следует учитывать, что нужно менять только количество товара и при обнаружении изменения товара нужно также генерировать сообщение об ошибке.

7.4 Запросы, учитывающие двойные связи

В нашем примере идет перемещение между двумя складами информация о которых содержится в одной таблиц. Рассмотрим как правильно построить запрос для выборки данных. Для начала вызовем редактор запроса в режиме дизайнера (нажмем кнопку **Новый запрос** на панели инструментов). После этого переместим в область дизайнера таблицы **storage_description** и **storage_transfers** (Рис. 7.19).

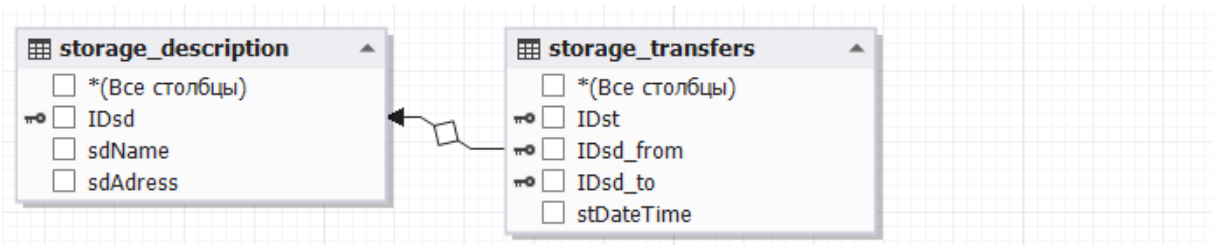


Рис. 7.19. Размещение таблиц в области дизайнера

Поскольку используются два склада то необходимо по каждому выводить информацию. Для этого добавим в область дизайнера еще один экземпляр таблицы **storage_description** (Рис. 7.20).

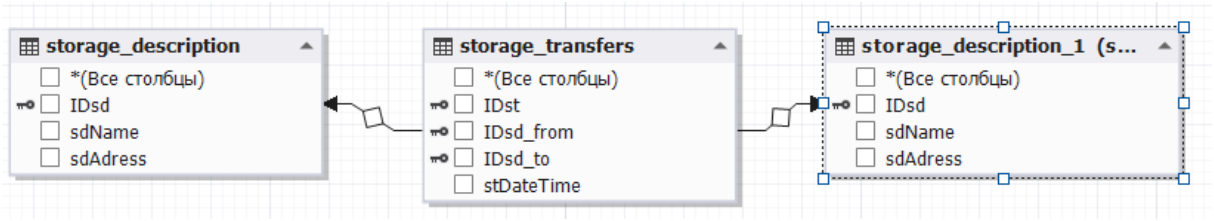


Рис. 7.20. Добавление второго экземпляра таблицы storage_description

Как можно заметить из рисунка по умолчанию связь идет через одно поле. В связи с этим необходимо удалить связь. Для этого воспользуемся контекстным меню (Рис. 7.21), в котором выберем пункт **Удалить из диаграммы**.

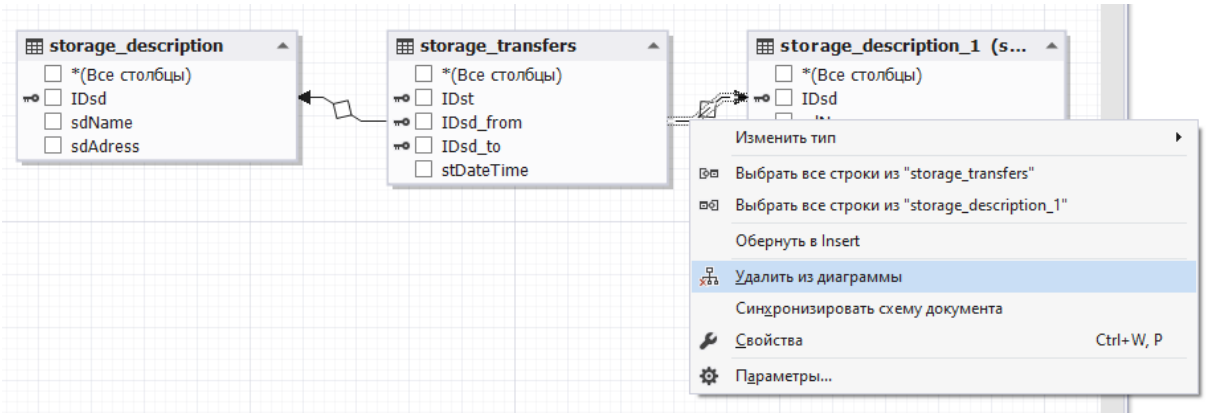


Рис. 7.21. Контекстное меню связи

После удаления связи необходимо соединить поле **IDsd** с **IDsd_to** зажав левую кнопку мыши на поле **IDsd** (Рис. 7.22).

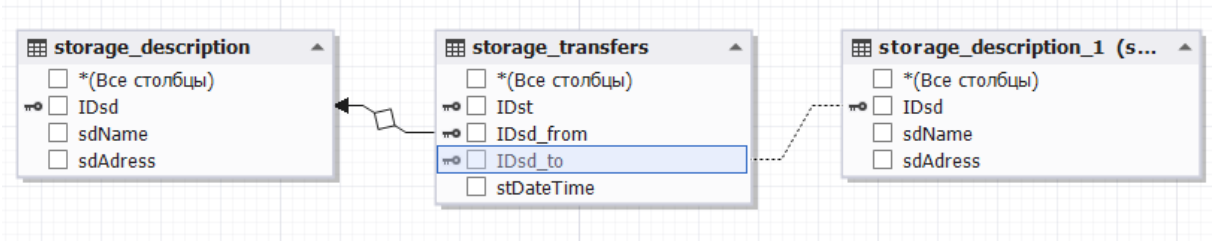


Рис. 7.22. Настройка связи

Можно выполнить соединение и в обратном порядке, для дизайнера запроса это не имеет значения (он сам определит направление связи по настроенному внешнему ключу). Теперь остается выполнить настройку отображения полей (помечаем галочками желаемые поля). Следует обратить внимание, что будут два поля с одинаковым названием **sdName**. Для того чтобы не было путаницы какое поле **sdName** отвечает за название склада отправления, а какое за склад получения товара добавим для них псевдонимы

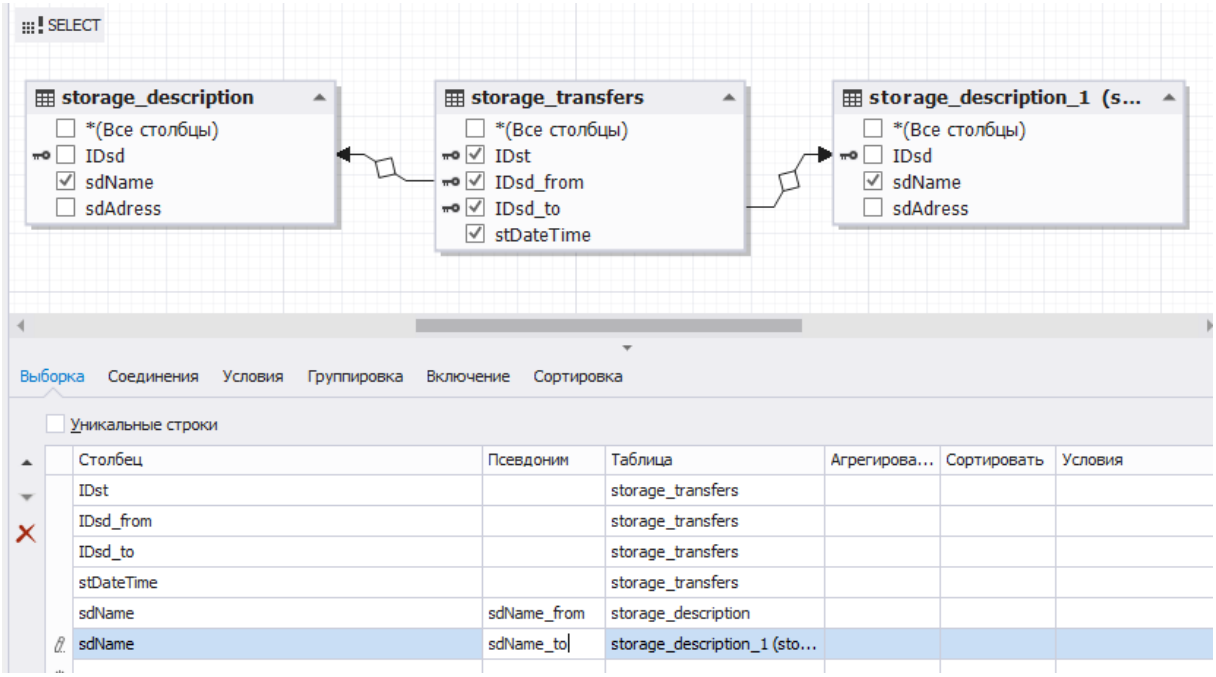


Рис. 7.22. Настройка выборки в дизайнере запроса

Полученный код запроса использовать для создания представления **view_storage_transfers**.

7.5 Использование хранимых процедур для выбора данных

Хранимые процедуры представляют собой набор команд языка SQL, которые могут компилироваться и храниться на сервере. Хранимые процедуры запускаются во всех средах. С момента создания хранимой процедуры не важно, какое приложение используется для вызова процедуры. Также не важно, на каком языке программирования написано это приложение, поскольку процедура находится на сервере БД и не зависит от окружения.

Создадим хранимую процедуру с именем **Get_Received_Goods**, которая будет выбирать поступившие на склад товары за определенный промежуток времени. Щелкнем правой кнопкой мыши в проводнике на подкаталоге **Процедуры** и выберем пункт меню **Новая процедура**, далее зададим имя процедуры, два параметра **DateStart**, **DateEnd** и зададим **SQL**-запрос на выборку данных:

```
SELECT operations.oDateTime, view_operation_goods_list.gcName,
  view_operation_goods_list.ocCount
FROM view_operation_goods_list
  INNER JOIN operations
    ON view_operation_goods_list.IDoperation = operations.IDoperation
WHERE operations.oIsSale = 0 AND CAST(operations.oDateTime AS DATE)
  BETWEEN DateStart AND DateEnd;
```

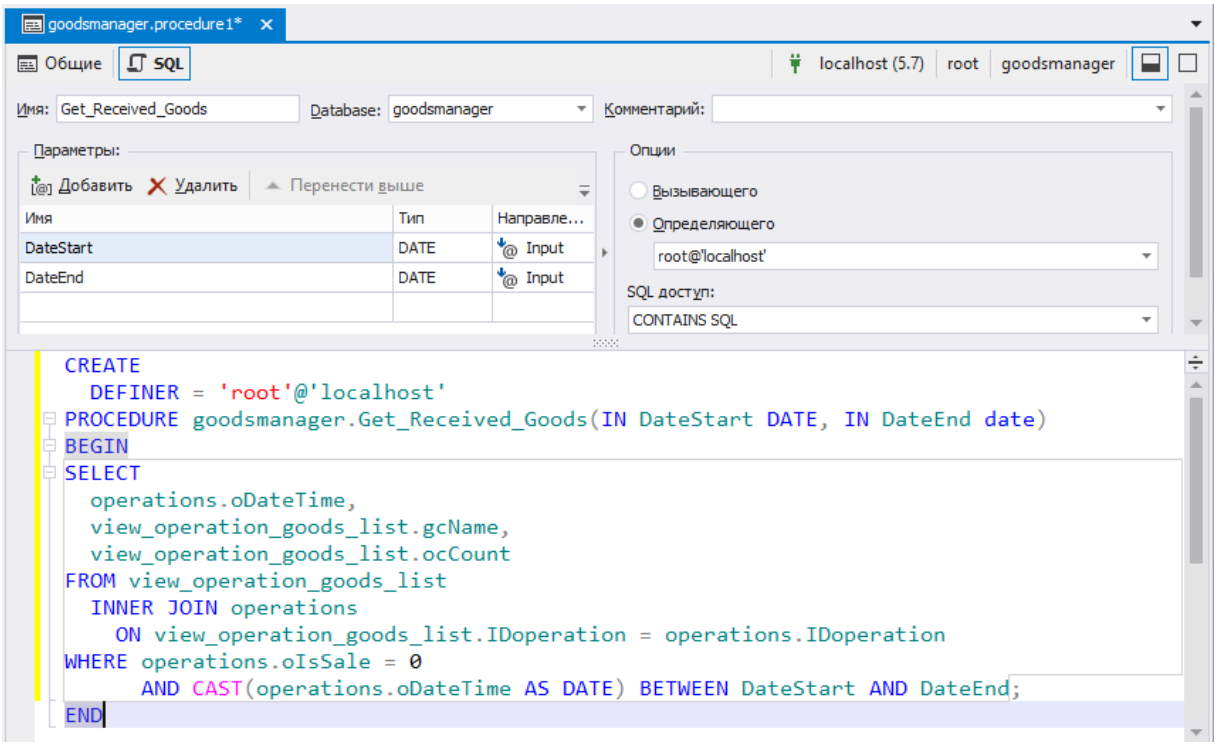


Рис. 7.22. Настройка выборки в дизайнере запроса

Реализуем обращение к созданной процедуре из клиентского приложения написанного на **Delphi**. Для этого создадим новый проект, организуем подключение к базе данных с использование компонентов **TADOConnection**, **ADOQuery, DataSource и TDBGrid.** Для настройки временного интервала для выборки данных разместим на форме два компонента **TDateTimePicker** и кнопку (**TButton),** для поясняющих сообщений будем использовать **Tlabel.** Пример размещения компонентов и настройки их визуальных свойств приведен на рисунке 7.23.

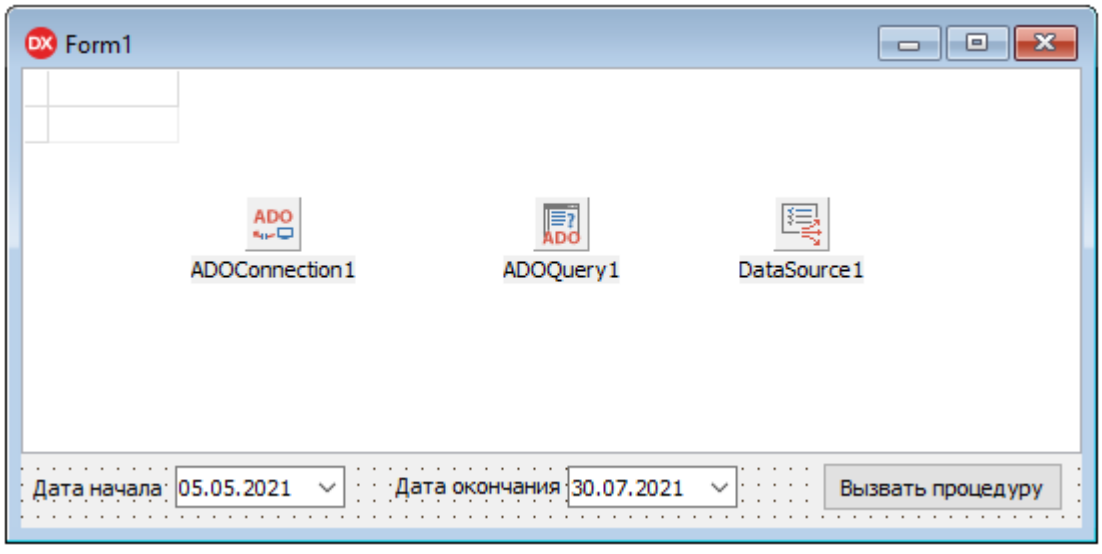


Рис. 7.23. Пример размещения компонентов

Хранимая процедура вызывается при помощи **SQL**-запроса с использованием ключевого слова **CALL**. Для того чтобы правильно выполнить запрос на вызов процедуры **Get_Received_Goods,** необходимо при помощи функции **FormatDateTime** привести сведения о дате из компонентов TDateTimePicker перевести в формат MySQL (дата в **MySQL** хранится в формате **YYYY-MM-DD)** Для компонента **Button1** запишем следующий обработчик нажатия:

```
procedure TForm1.Button1Click(Sender: TObject);
var
  SQLstr, DateB, DateE: AnsiString;
begin
  DateB := QuotedStr(FormatDateTime('YYYY-MM-DD',
    DateTimePicker1.Date));
  DateE := QuotedStr(FormatDateTime('YYYY-MM-DD',
    DateTimePicker2.Date));
  SQLstr:= 'CALL Get_Received_Goods(' + DateB + ', ' + DateE + ')';
  try
    ADOQuery1.Active := false;
    ADOQuery1.SQL.Clear;
    ADOQuery1.SQL.Add(SQLstr);
    ADOQuery1.Active := True;
  except
    on e: Exception do
  end;
end;
```

Пример работы вызова хранимой процедуры приведен на рис 7.24.

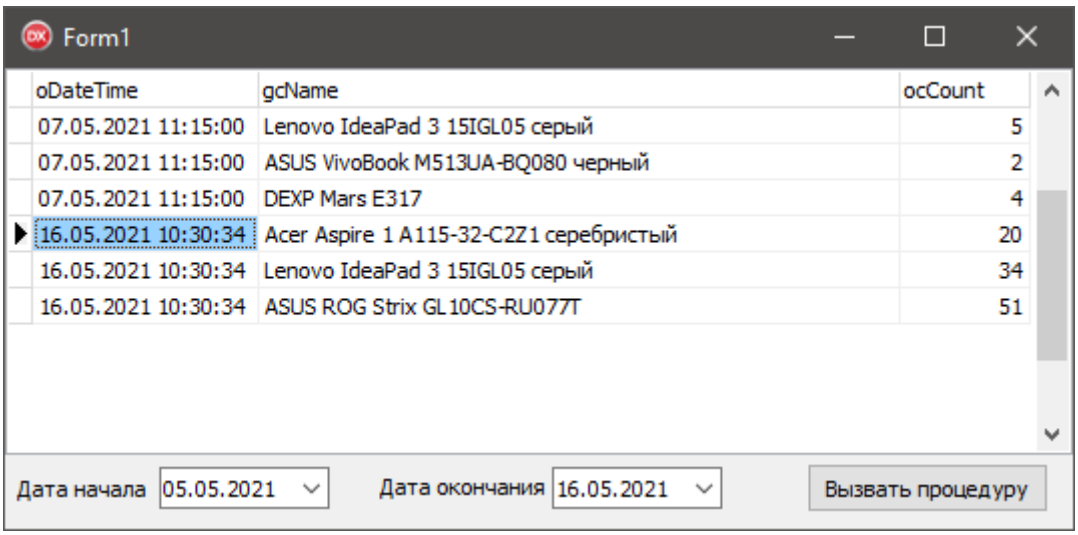


Рис. 7.24. Пример работы вызова процедуры

Используя хранимые процедуры можно осуществлять различную выборку данных, при этом возможна реализация, где один из параметров

сможет менять полностью структуру выборки на основе операторов ветвления и тем самым возвращать данные из различных таблиц.

7.6 Профилирование запросов

При работе с высоконагруженными проектами часто приходится заниматься оптимизацией запросов на низком уровне (менять порядок соединения, настраивать индексы ит.п.). Для определения высоконагруженных (ресурсоемких действий) можно воспользоваться возможностями профилировщика **SQL**-запросов.

Встроенный профилировщик программы **dbForge Studio** имеет следующие функции:

– визуальное профилирование запросов;

– оптимизацию **SQL**-запросов;

– сравнение результатов профилирования.

Создадим запрос, на котором проверим как работает профилировщик. Для демонстрации механизма профилирования отобразим количество перемещаемых товаров с группировкой по типу товара. Настройка выборки данных показана на рисунке 7.25, а параметры группировки для данного запроса на рисунке 7.26.

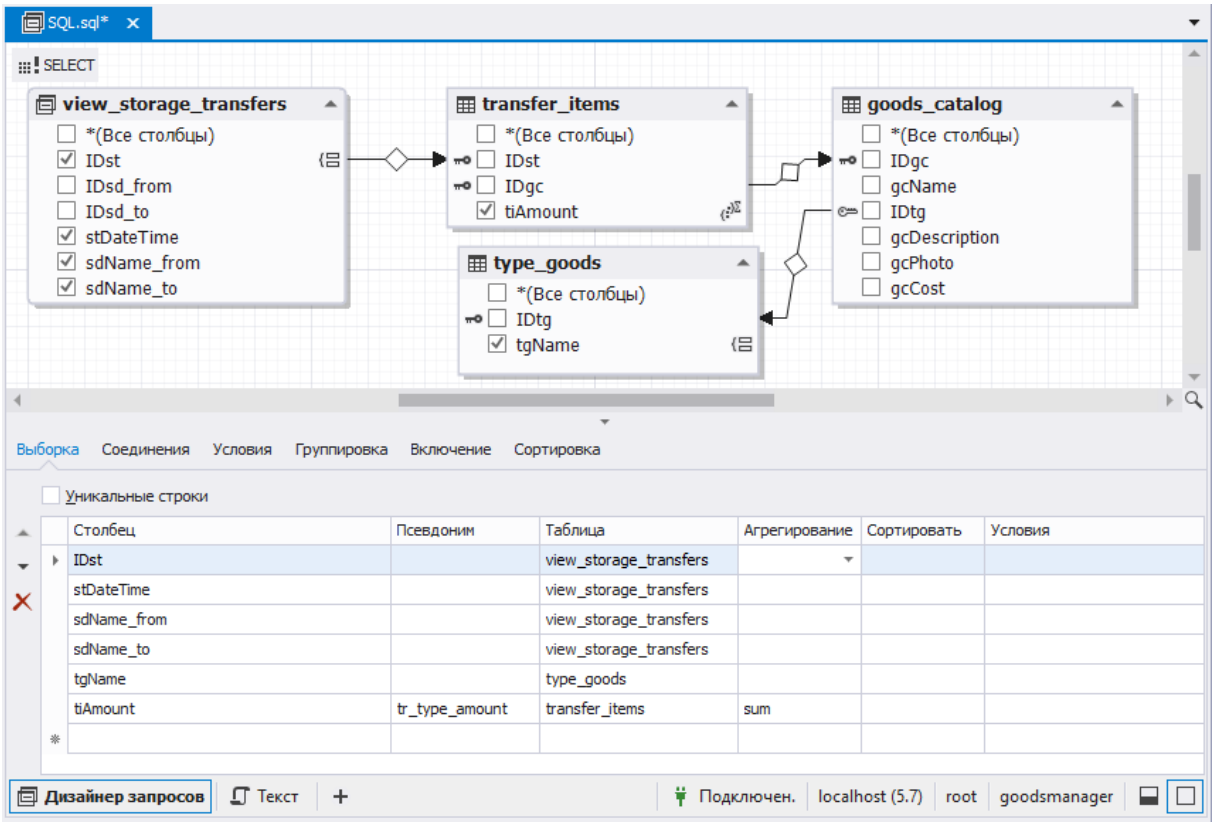


Рис. 7.25. Настройка выборки у запроса

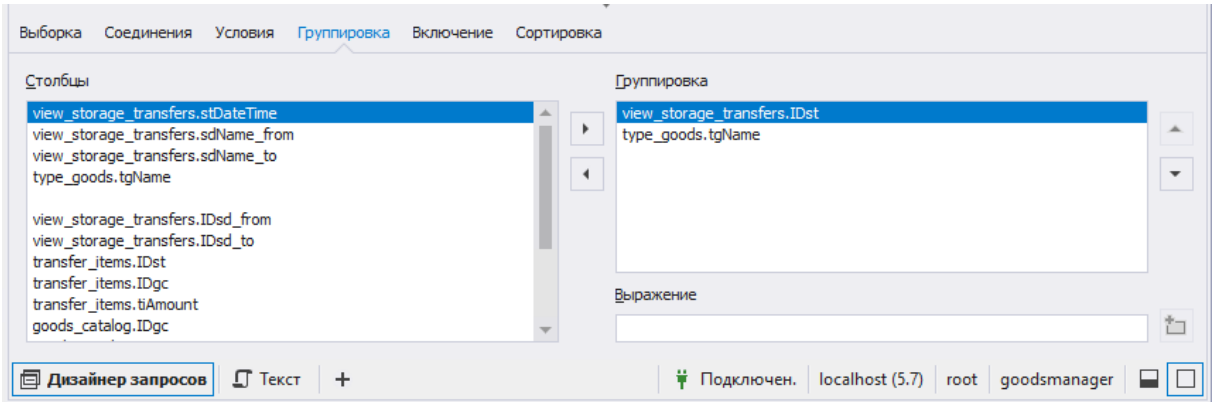


Рис. 7.26. Настройка параметров группировки для запроса

Чтобы просмотреть результаты профилирования, войдем в режим профилирования, выберем меню **SQL**, пункт **Режим профилирования** и выполним запрос, нажав кнопку **Выполнить**. В результате вы увидите статистическую сводку по результатам профилировщика (Рис 7.27).

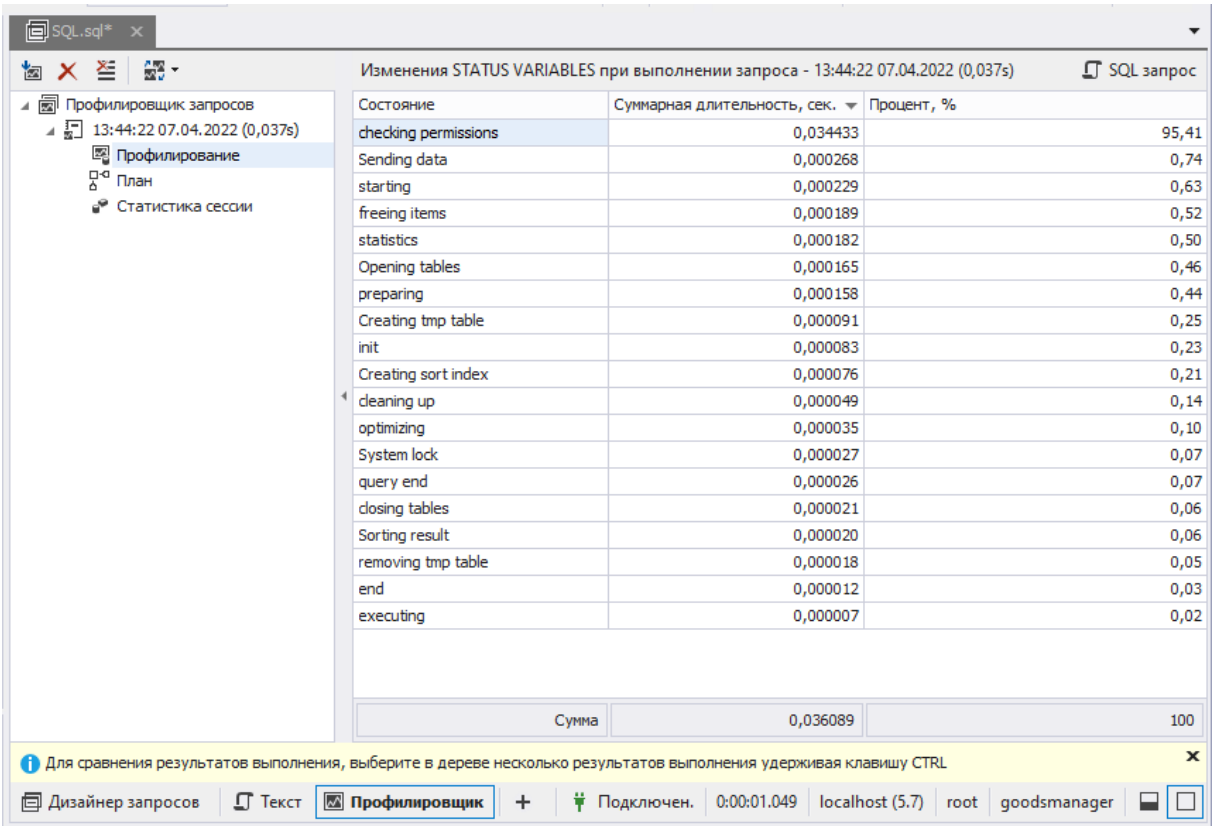


Рис. 7.27. Пример работы профилировщика запросов

После профилирования запроса можно просмотреть статистику сессии, которая позволяет определить самые ресурсоемкие части запроса.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Каким образом делаются двойные связи?
2. Как можно решить проблему реализации связи многие-ко-многим?
3. Какие особенности у создания внешних ключей при модификации таблицы с данными?
4. Каким образом можно организовать вставку данных из одной таблицы в другую?
5. Какая структура у запроса выборки данных в несколько пользовательских переменных?
6. Каким образом генерируютмя пользовательские ошибки в MySQL&
7. Охарактеризуйте оператор **SIGNAL** и его параметры**.**
8. Каким образом создаются хранимые функции? Приведите сферы применения хранимых функций.
9. Каким образом формируются запросы на выборку данных при наличии двойной связи между таблицами?
10. Охарактеризуйте хранимые процедуры.
11. В чем состоит отличие хранимой процедуры от функции?
12. Как вызываются хранимые процедуры?
13. Для чего нужно профилирование запросов?