


Кафедра информационных систем и цифровых технологий

 \_\_\_\_\_ Руководитель

« ИЗ » ОС 20 И Г.

сн замочили

по дисциплине «Базы данных»

на тему: «Разработка БД для учёта продаж в магазине спортивной обуви»

Студент Жозеф В.

Шифр 190362

Институт приборостроения, автоматизации и информационных технологий

Направление подготовки 09.03.04 «Программная инженерия»

Группа 92ПГ

Руководитель \_\_\_\_\_ РЫЖЕНКОВ Д.В.

Оценка: « отт »


Дата 08.06.2021.

Орел 2021

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ  
РОССИЙСКОЙ ФЕДЕРАЦИИ  
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ  
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ  
«ОРЛОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ  
ИМЕНИ И.С. ТУРГЕНЕВА»

Кафедра информационных систем и цифровых технологий

УТВЕРЖДАЮ:

 Зав. кафедрой

«11» февраля 2021 г.

**ЗАДАНИЕ**

**на курсовую работу**

по дисциплине «Базы данных»

Студент Жозеф В.

Шифр 190362

Институт приборостроения, автоматизации и информационных технологий

Направление подготовки 09.03.04 «Программная инженерия»

Группа 92ПГ

1 Тема курсовой работы

«Разработка БД для учёта продаж в магазине спортивной обуви»

2 Срок сдачи студентом законченной работы «8» июня 2021

### 3 Исходные данные

Предметная область по учету продаж в магазине спортивной обуви

### 4 Содержание курсовой работы

Анализ предметной области по учету продаж в магазине спортивной обуви

Проектирование базы данных для учета продаж в магазине спортивной обуви

Реализация базы данных для учета продаж в магазине спортивной обуви

### 5 Отчетный материал курсовой работы

Пояснительная записка курсовой работы

Руководитель \_\_\_\_\_ Рыженков Д.В.

Задание принял к исполнению: «11» февраля 2021

подпись студента \_\_\_\_\_



## Содержание

Введение.....	4
1 Описание предметной области .....	5
2 Проектирование базы данных.....	6
2.1 Разработка концептуальной схемы .....	6
2.2 Разработка логической схемы базы данных.....	9
2.3 Разработка физической схемы базы данных .....	13
3 Реализация базы данных.....	17
3.1 Скрипты создания базы данных .....	17
3.2 Скрипты заполнения базы данных .....	20
3.3 Реализация запросов базы данных .....	21
3.4 Разработка процедур .....	23
3.5 Разработка триггеров .....	24
Заключение .....	27
Список литературы .....	28
Приложение А (обязательное) .....	29
Приложение Б (обязательное).....	37
Приложение В (обязательное) .....	42
Приложение Г (обязательное).....	43

## **Введение**

В наше время все автоматизировано. Технология создала себе место во всех сферах. Поэтому для управления информацией, быстрого доступа к ней и уменьшения рабочей нагрузки большинство магазинов используют автоматизированную информационную систему для своих баз данных.

база данных — это совокупность взаимосвязанных данных, которые организованы по определенным правилам и относятся к некоторой предметной области или ее части.

Система управления базами данных (СУБД) — совокупность программных и лингвистических средств общего или специального назначения, обеспечивающих управление созданием и использованием баз данных

Целью данной курсовой работы является разработка базы данных для учета продаж в магазине спортивной обуви, которая позволит не только систематизировать огромное количество данных, но и обеспечит быстрый доступ пользователю к ним.

Чтобы достичь этой цели, необходимо выполнить следующие задачи:

- описать предметную область;
- разработать концептуальную схему базы данных;
- разработать логическую схему базы данных;
- разработать физическую схему базы данных;
- реализовать базу данных;
- реализовать запросы базы данных.

## **1 Описание предметной области**

Магазин специализируется на продаже обуви для различных видов спорта: футбола, баскетбола, фитнеса и т. Д. В магазине представлен большой ассортимент спортивной обуви. Когда покупатель приходит в магазин, он может выбрать обувь по разным характеристикам. Обувь классифицируется по категории (виду спорта), материалу, производителю, размеру, цвету, типу (женских, мужских или детских) и количеству, доступному в магазине.

Покупатель имеет возможность совершать покупки удаленно. Для этого ему необходимо позвонить в магазин, спросить у продавца необходимую ему информацию. Если в наличии есть кроссовки, которые он хочет, он может зарезервировать их для себя, предоставив продавцу свою личную информацию, такую как его имя, Фамилия, номер телефона, а также информацию о выбранных кроссовках: производитель, цвет, размер, тип (женских, мужских или детских), количество. Продавец сообщит ему общую цену, которую он должен будет заплатить, код для его заказа и крайний срок, когда нужно прийти и забрать кроссовки. Но покупатель также имеет возможность приобрести товар по своему выбору непосредственно в магазине, если он доступен.

После оплаты выбранного товара кассир должен предоставить покупателю квитанцию, содержащую следующую информацию: название магазина, адрес, имя кассира, купленные кроссовки, цена за единицу, итоговая цена, количество и гарантийный срок.

База данных учтет все элементы, которые только что были упомянуты. Это позволит оптимизировать и автоматизировать продажи в магазине.

## **2 Проектирование базы данных**

### **2.1 Разработка концептуальной схемы**

Концептуальное моделирование данных — это первоначальный этап разработки проекта постоянных данных и хранилища постоянных данных для системы. Во многих случаях постоянные данные для системы управляются системой управления реляционной базой данных. Сущности, определенные на концептуальном уровне в моделях и системных требованиях, будут развиты с помощью задач анализа вариантов использования, проекта вариантов использования и проекта базы данных в детальный проект физических таблиц, которые будут применены в системе управления базой данных.

Концептуальное проектирование базы данных абсолютно не зависит от таких подробностей ее реализации, как тип выбранной целевой СУБД, набор создаваемых прикладных программ, используемые языки программирования, тип выбранной вычислительной платформы, а также от любых других особенностей физической реализации. При разработке концептуальная модель данных постоянно подвергается тестированию и проверке на соответствие требованиям пользователей. Созданная концептуальная модель данных предприятия является источником информации для этапа логического проектирования базы данных.

Для единообразия программирования баз данных введены следующие понятия для концептуальных баз данных:

**Объект или сущность.** Это фактическая вещь или объект (для людей) за которой пользователь (заказчик) хочет наблюдать. Например, Иванов Иван Иванович;

**Атрибут** — это характеристика объекта, соответствующая его сущности.

Третье понятие в проектировании концептуальной базы данных это связь или отношения между объектами.

В нашем случае концептуальная модель будет представлять следующие отношения:

- Заказ с атрибутами: Код заказа, Дата заказа, Срок, Состояние заказа, итоговая цена, Количество товара;
- Магазин с атрибутами: Код заказа, название магазина, адрес магазина;
- Продажа с атрибутами: Код продажи, гарантийный срок, итоговая цена, количество товара, дата продажи;
- Продавец с атрибутами: Имя, фамилия, отчество, код продавца;
- Категория с атрибутами: Название, код категории;
- Фирма производитель с атрибутами: Код фирма, название;
- Материал с атрибутами: Название материала, код материала;
- Размер с атрибутами: Код размера, значение размера, количество;
- Цвет с атрибутами: Код цвета, название цвета, количество;
- Кроссовки с атрибутами: Код кроссовки, название товара, количество, цена;
- Клиенты с атрибутами: Имя, фамилия, отчество, код клиента, номер телефона;
- Модель с атрибутами: Код модели, количество.

Также на концептуальной схеме указаны типы связей между таблицами, а в ромбах – взаимодействие таблиц между собой.

Есть разные виды связей:

- Один к одному: Связь один к одному образуется, когда ключевой столбец (идентификатор) присутствует в другой таблице, в которой тоже является ключом либо свойствами столбца задана его уникальность;



- Один ко многим: в типе связей один ко многим одной записи первой таблицы соответствует несколько записей в другой таблице;
- Многие ко многим: если нескольким записям из одной таблицы соответствует несколько записей из другой таблицы, то такая связь называется «многие ко многим» и организовывается посредством связывающей таблицы.

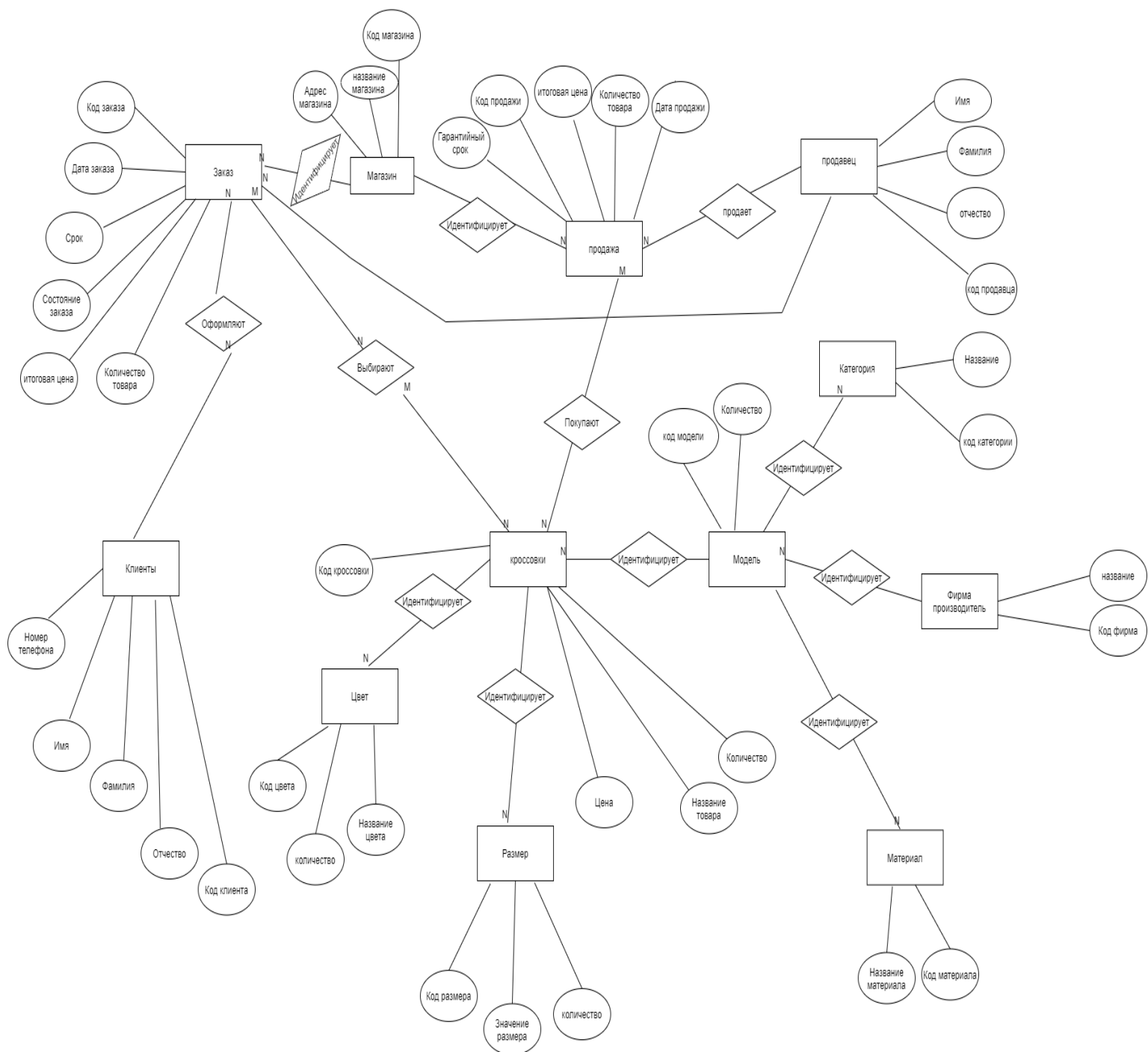


Рисунок 1 – Концептуальная схема базы данных

## 2.2 Разработка логической схемы базы данных

Логическая модель данных - описание объектов предметной области, их атрибутов и взаимосвязей между ними в том объеме, в котором они подлежат непосредственному хранению в базе данных системы.

Логическая модель строится в несколько этапов с постепенным приближением к оптимальному для данных условий варианту. Эффективность такой модели зависит от того, насколько близко она отображает изучаемую предметную область. К предметной области относятся объекты (документы, счета, операции над ними и пр.), а также характеристики данных объектов, их свойства, взаимодействие и взаимное влияние.

Таким образом, при построении логической модели данных сначала выявляются те объекты, которые интересуют пользователей проектируемой базы данных. Затем для каждого объекта формулируются характеристики и свойства, достаточно полно описывающие данный объект. Эти характеристики в дальнейшем будут отражены в базе данных как соответствующие поля.

Логическая модель данных строится в рамках одного из трех подходов к созданию баз данных. Выделяют следующие виды логических моделей базы данных:

- иерархическая;
- сетевая;
- реляционная.

В реализованной нами логической модели все таблицы находятся третьей нормальной форму.

Требование третьей нормальной формы (3NF) заключается в том, чтобы в таблицах отсутствовала транзитивная зависимость.

На основе концептуальной схемы, основными сущностями являются:

- Заказ;
- Магазин;

- Продажа;
- Продавец;
- Категория;
- Фирма;
- Материал;
- Размер;
- Заказ\_has\_Кроссовки;
- Продажа\_has\_Кроссовки;
- Модель\_has\_Материал;
- Цвет;
- Кроссовки;
- Клиенты;
- Модель.

В логической модели массивы связаны между собой, так:

- Сущность «Продавец» связывает сущности «Продажа» и «Заказ» Один продавец может оформлять много заказов или продаж. Так что нужно связь «один ко многим»;
- Сущность «Кроссовки» связывает сущности «Цвет», «Размер», «Модель». Один экземпляр кроссовки может имеет разные цветов, также один экземпляр кроссовки может иметь несколько размеров или модель. Сущность кроссовки имеет связь «один ко многим» с этими сущностями;
- Сущность «Категория» связывает сущности «Модель». Между ними есть связь «один ко многим». Одна категория может иметь несколько модели;
- Сущность «Фирма» связывает сущности «Модель». Между ними есть связь «один ко многим». Одна фирма может иметь несколько модели;
- Сущность «Клиент» связывает сущности «Заказ» Один заказ может имеет один заказчик (клиент) но один заказчик может сделать несколько заказов. У них связь «один ко многим»;

- Сущность «Магазин» связывает сущности «Продажа» и «Заказ» В одном магазине делается много продаж и заказов так что связь между ними «один ко многим»;

Все вышеперечисленные таблицы реализуют связи «один ко многим» между сущностями.

в следующих таблицах реализована связь «многие-ко-многим»:

- Сущность «Кроссовки» связывает сущности «Заказ» и приведет к созданию таблицы «Заказ\_has\_Кроссовки». Заказ может содержать много кроссовок и один экземпляр кроссовок может быть в разные заказы. Между ними есть связь «многие-ко-многим»;
- Сущность «Кроссовки» связывает сущности «Продажа» и приведет к созданию таблицы «Продажа\_has\_Кроссовки». Продажа может содержать много кроссовок и один экземпляр кроссовок может быть в разные продажи. Между ними есть связь «многие-ко-многим»;
- Сущность «Материал» связывает сущности «Модель» и приведет к созданию таблицы «Модель\_has\_Материал». Один материал может быть в несколько кроссовок и один экземпляр кроссовок может быть сделан разными материалами. Между ними есть связь «многие-ко-многим»;

Логическая диаграмма, которую мы реализовали, находится в третьей нормальной форме. Отношение находится в третьей нормальной форме (3НФ), если оно удовлетворяет определению 2НФ и ни один из его не ключевых атрибутов не зависит функционально от любого другого не ключевого атрибута. Иными словами, не ключевые столбцы не должны пытаться играть роль ключа в таблице, т.е. они действительно должны быть не ключевыми столбцами, такие столбцы не дают возможности получить данные из других столбцов, они дают возможность посмотреть на информацию, которая в них содержится, так как в этом их назначение. Например: сущность «Цвет», «Размер», «Материал», «Категория». затем, чтобы подтвердить

третью нормальную форму, мы должны доказать, что схема находится во второй нормальной форме.

Чтобы база данных находилась во второй нормальной форме (2NF), необходимо чтобы ее таблицы удовлетворяли следующим требованиям:

- таблица должна находиться в первой нормальной форме, таблица должна иметь ключ,
- все не ключевые столбцы таблицы должны зависеть от полного ключа (в случае если он составной).

Если ключ составной, т.е. состоит из нескольких столбцов, то все остальные не ключевые столбцы должны зависеть от всего ключа, т.е. от всех столбцов в этом ключе. Если какой-то атрибут (столбец) зависит только от одного столбца в ключе, значит, база данных не находится во второй нормальной форме. Иными словами, в таблице не должно быть данных, которые можно получить, зная только половину ключа, т.е. только один столбец из составного ключа. Например: «Заказ\_has\_Кроссовки» её атрибут «КолКрЗаказ» зависит не только от `заказ_idЗаказ` но и от `Кроссовки_idКроссовки`. Теперь мы можем доказать принадлежность схемы к первой нормальной форме. Чтобы база данных находилась в 1 нормальной форме, необходимо чтобы ее таблицы соблюдали следующие реляционные принципы:

- В таблице не должно быть дублирующих строк;
- В каждой ячейке таблицы хранится атомарное значение;
- В столбце хранятся данные одного типа;
- Отсутствуют массивы и списки в любом виде.

Все наши сущности соблюдают эти правила.

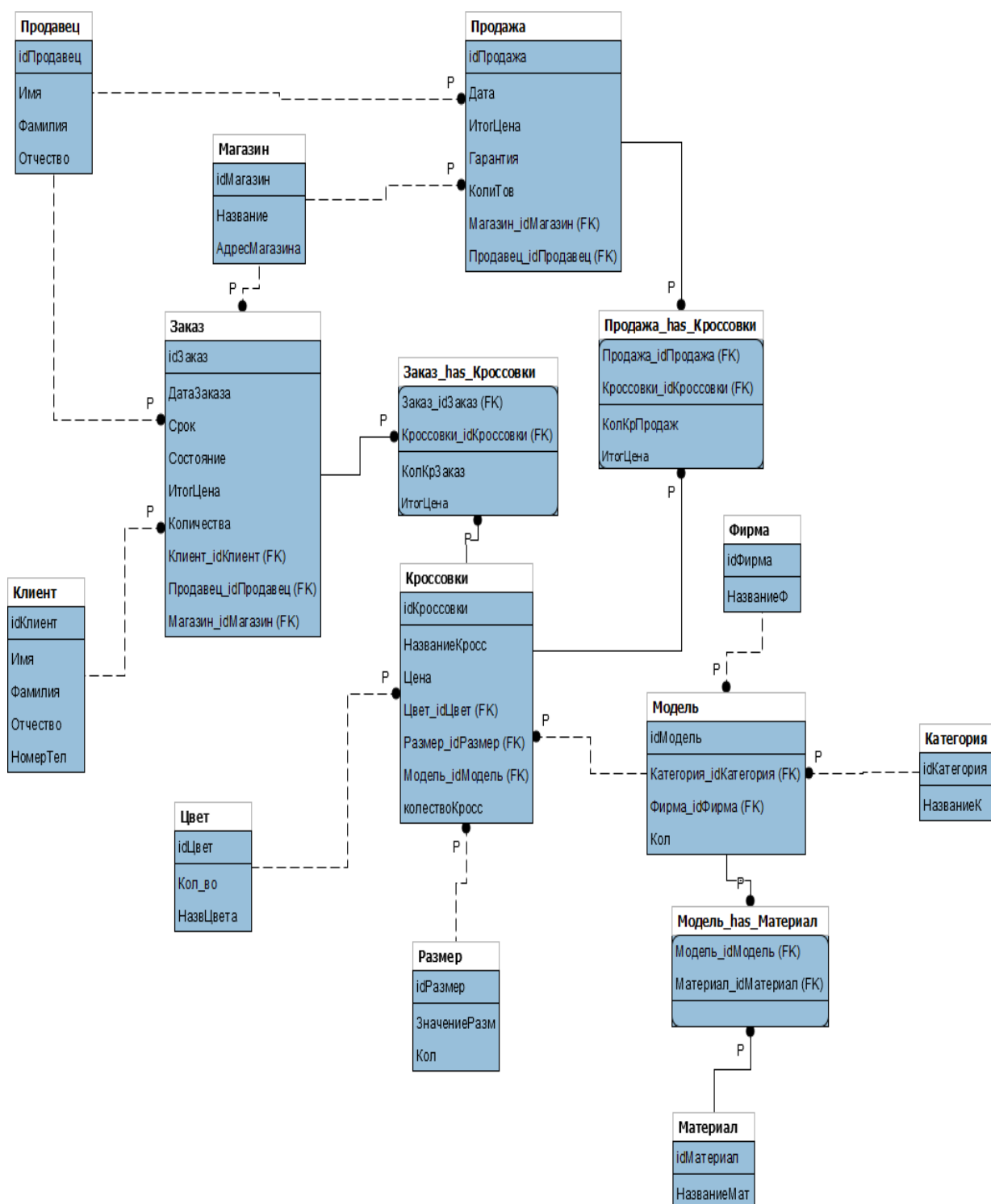


Рисунок 2 – Логическая схема базы данных

### 2.3 Разработка физической схемы базы данных

Для привязки логической модели к среде хранения используется модель данных физического уровня, для краткости часто называемая



физической моделью. Эта модель определяет способы физической организации данных в среде хранения. Модель физического уровня строится с учетом возможностей, предоставляемых СУБД. Описание физической структуры базы данных называется схемой хранения. Соответствующий этап проектирования БД называется физическим проектированием. СУБД обладают разными возможностями по физической организации данных. В связи с этим различаются для конкретных систем сложность и трудоемкость физического проектирования, и набор выполняемых шагов.

Физическая модель данных описывает данные средствами конкретной СУБД. На этапе физического проектирования учитывается специфика конкретной модели данных и специфика конкретной СУБД. Отношения, разработанные на стадии формирования логической модели данных, преобразуются в таблицы, атрибуты становятся столбцами таблиц, для ключевых атрибутов создаются уникальные индексы, домены преобразуются в типы данных, принятые в используемой СУБД. Ограничения, имеющиеся в логической модели данных, реализуются различными средствами СУБД, например, при помощи индексов, ограничений целостности, триггеров, хранимых процедур. При этом решения, принятые на уровне логического моделирования, определяют некоторые границы, в пределах которых можно развивать физическую модель данных. Точно также в пределах этих границ можно принимать различные решения. Например, отношения, содержащиеся в логической модели данных, должны быть преобразованы в таблицы, но для каждой таблицы можно дополнительно объявить различные индексы, повышающие скорость обращения к данным. Многое тут зависит от конкретной СУБД.

Физическое проектирование БД предполагает также выбор эффективного размещения БД на внешних носителях для обеспечения наиболее эффективной работы приложения.

Как правило, основной целью физического проектирования базы данных является описание способа физической реализации логического проекта базы данных.

На физической схеме также присутствуют идентифицирующие и не идентифицирующие связи между таблицами.

В проектируемой таблице рассматриваются следующие типы данных:

VARCHAR () – символьный тип;

DATE – тип даты;

INT – целый тип;

DECIMAL– типы числовых данных с фиксированными точностью и масштабом.

Бизнес-правила представляют собой конкретные требования и условия для функций, задающие поведение данных. В практике проектирования бизнес-правила используются для поддержки целостности данных в базе данных.

Определим некоторые бизнес правил для нашей базы данных:

- 1) Цена кроссовок не должна быть меньше или равна 0.
- 2) Общая цена экземпляра кроссовок в продаже должна быть равна произведению цены единицы экземпляра на количество проданных экземпляров.
- 3) Общая цена кроссовок в заказе или продаже должна быть суммой цен всех кроссовок, включенных в одну продажу или в один заказ.

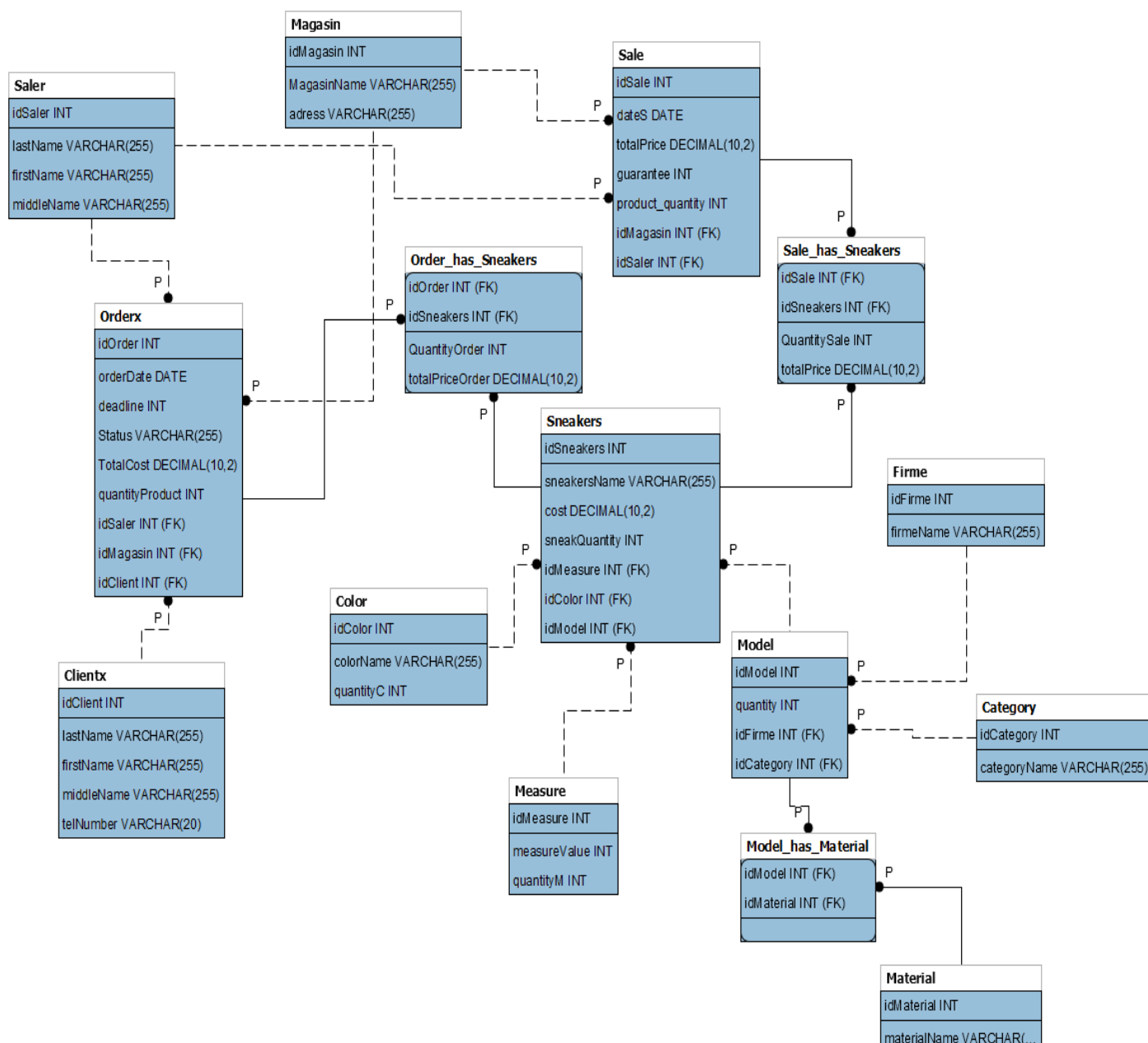


Рисунок 3 – Физическая схема базы данных

### 3 Реализация базы данных

#### 3.1 Скрипты создания базы данных

Создание таблицы БД состоит из двух этапов. На первом этапе определяется ее структура: состав полей, их имена, последовательность размещения в таблице, тип данных, размер, ключевые поля и другие свойства полей. На втором этапе производится создание записей таблицы и заполнение их данными. Ниже представлен скрипт создания таблицы «Sneakers»

```
CREATE TABLE IF NOT EXISTS `shoesSport`.`Sneakers` (
  `idSneakers` INT NOT NULL AUTO_INCREMENT,
  `sneakersName` VARCHAR (255) NOT NULL,
  `cost` DECIMAL(10,2) NOT NULL,
  `sneakQuantity` INT NOT NULL,
  `idMeasure` INT NOT NULL,
  `idColor` INT NOT NULL,
  `idModel` INT NOT NULL,
  PRIMARY KEY (`idSneakers`),
  CONSTRAINT `fk_idModel_2x`
  FOREIGN KEY (`idModel`)
  REFERENCES `shoesSport`.`Model` (`idModel`)
  ON DELETE CASCADE
  ON UPDATE CASCADE,
  CONSTRAINT `fk_idMeasure_1x`
  FOREIGN KEY (`idMeasure`)
  REFERENCES `shoesSport`.`Measure` (`idMeasure`)
  ON DELETE CASCADE
  ON UPDATE CASCADE,
  CONSTRAINT `fk_idColor_1x` FOREIGN KEY (`idColor`)
  REFERENCES `shoesSport`.`Color` (`idColor`)
  ON DELETE CASCADE
```

ON UPDATE CASCADE)

ENGINE = InnoDB;

Инструкция CREATE TABLE используется для создания таблицы в MySQL.

Тип данных определяет тип данных, которые может содержать столбец. В нашем случае мы использовали следующие типы: int, decimal, varchar.

После того как тип данных, можно указать другие необязательные атрибуты для каждого столбца:

NOT NULL - Каждая строка должна содержать значение для этого столбца, значения NULL не допускаются

DEFAULT value - Установите значение по умолчанию, которое добавляется, когда не передается другое значение

UNSIGNED - Используется для числовых типов, ограничивает хранимые данные положительными числами и нулем

AUTO INCREMENT - MySQL автоматически увеличивает значение поля на 1 каждый раз при добавлении новой записи

PRIMARY KEY - Используется для уникальной идентификации строк в таблице. Столбец с параметром первичного ключа часто является идентификационным номером и часто используется с «auto increment»

Для того чтобы данные таблиц можно было однозначно идентифицировать, необходимо позаботиться об установке уникальных

первичных ключей. Для связи различных отношений базы данных необходимо установить внешние ключи и наложить на них ограничения для

сохранения целостности базы данных. Для примера представлен скрипт

установки внешнего ключа для атрибута «idModel» в сущности «Sneakers».

```

CONSTRAINT `fk_idModel_2x`
FOREIGN KEY (`idModel`)
REFERENCES `shoesSport`.`Model` (`idModel`)
ON DELETE CASCADE
ON UPDATE CASCADE,

```

С помощью ключевого слова **CONSTRAINT** можно задать имя для ограничений. В качестве ограничений могут использоваться **PRIMARY KEY**, **UNIQUE**, **DEFAULT**, **CHECK**. Ограничения могут носить произвольные названия, но, как правило, для применяются следующие префиксы:

"PK\_" - для **PRIMARY KEY**

"FK\_" - для **FOREIGN KEY**

"CK\_" - для **CHECK**

"UQ\_" - для **UNIQUE**

"DF\_" - для **DEFAULT**

В принципе необязательно задавать имена ограничений, при установке соответствующих атрибутов SQL Server автоматически определяет их имена. Но, зная имя ограничения, мы можем к нему обращаться, например, для его удаления. **CONSTRAINT имя\_ограничения** – необязательное имя для ограничения столбца или таблицы. Если ограничение нарушено, имя ограничения присутствует в сообщениях об ошибках, поэтому имена ограничений должны быть положительными, могут использоваться для передачи полезной информации об ограничениях клиентским приложениям.

**FOREIGN KEY ()** – указывается столбец таблицы, который будет представляет внешний ключ;

**REFERENCES ()** – указывается имя связанной таблицы, а затем в скобках имя связанного столбца, на который будет указывать внешний ключ.

Действие **ON DELETE CASCADE** позволяет удалять, но не обновлять данные родительского ключа, на которые имеются ссылки из дочерней таблицы. Когда данные в родительском ключе удаляются, все строки в



дочерней таблице, которые зависят от значений удаленного родительского ключа, также удаляются. Чтобы задать это ссылочное действие, включите параметр ON DELETE CASCADE в определение ограничения FOREIGN KEY.

NO ACTION - операция удаления строки из родительской таблицы отменяется. Именно это значение используется по умолчанию в тех случаях, когда в описании внешнего ключа фраза ON DELETE опущена.

Те же самые правила применяются в языке SQL и тогда, когда значение потенциального ключа родительской таблицы обновляется.

SET NULL - выполняется удаление строки из родительской таблицы, а во внешние ключи всех ссылающихся на нее строк дочерней таблицы записывается значение NULL ;

SET DEFAULT - выполняется удаление строки из родительской таблицы, а во внешние ключи всех ссылающихся на нее строк дочерней таблицы заносится значение, принимаемое по умолчанию;

### 3.2 Скрипты заполнения базы данных

Для ввода данных в БД понадобится команда INSERT INTO. Также важно знать название и тип данных полей (колонок) таблицы, которые мы заполняем. Ниже предоставлен пример заполнения таблицы «Sneakers»:

```
use shoesSport;
show columns from Sneakers;

insert                                into
Sneakers(idSneakers,sneakersName,cost,sneakquantity,idMeasure,      idColor,
idModel)values
(1, 'Speedy',3000, 100,1, 1,1 ),
(2, 'Mercurial',3500, 100,2, 2,2),
(3, 'Predator',4500, 100,3, 3,3),
```

```
(4, 'Danilo',3500, 100,2, 3,2),
(5, 'Sternilo',6000, 100,3, 1,2),
(6, 'Moizo',6000, 150,3, 2,2);
```

Оператор INSERT вставляет новые записи в таблицу. При этом значения столбцов могут представлять собой литеральные константы, либо являться результатом выполнения подзапроса. В первом случае для вставки каждой строки используется отдельный оператор INSERT; во втором случае будет вставлено столько строк, сколько возвращается подзапросом.

Список столбцов не является обязательным. В том случае, если он отсутствует, список вставляемых значений должен быть полный, то есть обеспечивать значения для всех столбцов таблицы. При этом порядок значений должен соответствовать порядку, заданному оператором CREATE TABLE для таблицы, в которую вставляются строки. Кроме того, эти значения должны относиться к тому же типу данных, что и столбцы, в которые они вносятся.

### 3.3 Реализация запросов базы данных

Язык SQL — это в первую очередь язык запросов, а кроме того он очень похож на естественный язык.

Каждый раз, когда требуется прочитать или записать любую информацию в БД, требуется составить корректный запрос. Такой запрос должен быть выражен в терминах SQL. На языке SQL выражаются все действия, которые можно провести с данными: от записи и чтения данных, до администрирования самого сервера СУБД.

В нашей базе данных один из запросов имеет следующий вид:

```
use shoesSport;

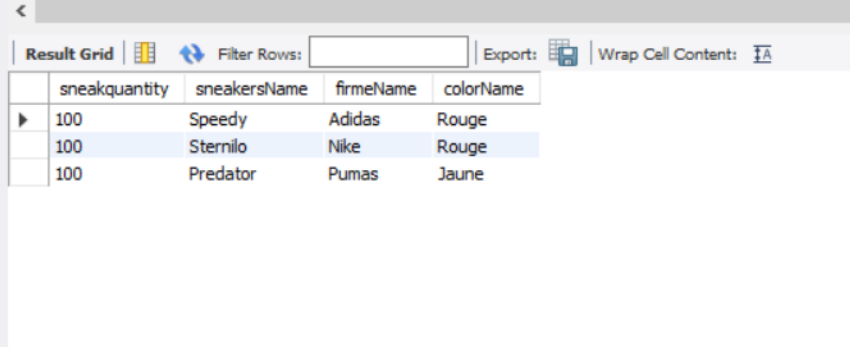
select Sneakers.sneakquantity, Sneakers.sneakersName, Firme.firmeName,
Color.colorName
```

```

from (((Sneakers
inner join Color on Sneakers.idColor = Color.idColor)
inner join Model on Sneakers.idModel = Model.idModel)
inner join Firme on Model.idFirme= Firme.idFirme)
group by Sneakers.idModel;

```

Этот запрос позволяет нам выбирать столбцы «sneakquantity», «sneakersName», «firmeName», «colorName» и отображать их вместе, соблюдая их соответствие. Для этого мы использовали ключевое слово «INNER JOIN»



	sneakquantity	sneakersName	firmeName	colorName
▶	100	Speedy	Adidas	Rouge
	100	Sternilo	Nike	Rouge
	100	Predator	Pumas	Jaune

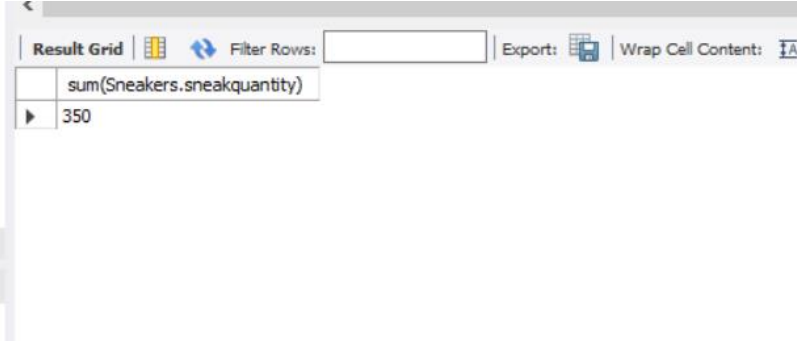
Рисунок 4 – Результат выполнения запроса

Ниже предоставлен второй пример запроса.

```

use shoesSport;
select sum(Sneakers.sneakquantity)
from Sneakers
where Sneakers.idMeasure=(select Measure.idMeasure from Measure
where measureValue=43);

```



	sum(Sneakers.sneakquantity)
▶	350

Рисунок 5– Результат выполнения второго запроса

Этот запрос позволяет нам получить количество обуви 43-го размера. SQL функция SUM используется для возврата суммы выражения в операторе SELECT.

### 3.4 Разработка процедур

Хранимые процедуры – это объекты базы данных, в которых заложен алгоритм в виде набора SQL инструкций. Иными словами, можно сказать, что хранимые процедуры – это программы внутри базы данных. Хранимые процедуры используются для сохранения на сервере повторно используемого кода, например, Вы написали некий алгоритм, последовательный расчет или многошаговую SQL инструкцию, и чтобы каждый раз не выполнять все инструкции, входящие в данный алгоритм, Вы можете оформить его в виде хранимой процедуры. При этом, когда Вы создаете процедуру SQL, сервер компилирует код, а потом, при каждом запуске этой процедуры SQL сервер уже не будет повторно его компилировать.

Для создания процедуры необходимо воспользоваться оператором CREATE PROCEDURE. После этого следует имя данной процедуры, а затем непосредственно тело процедуры.

Для того чтобы запустить хранимую процедуру в SQL Server, необходимо перед ее названием написать команду EXECUTE, также возможно сокращенное написание данной команды EXEC или CALL.

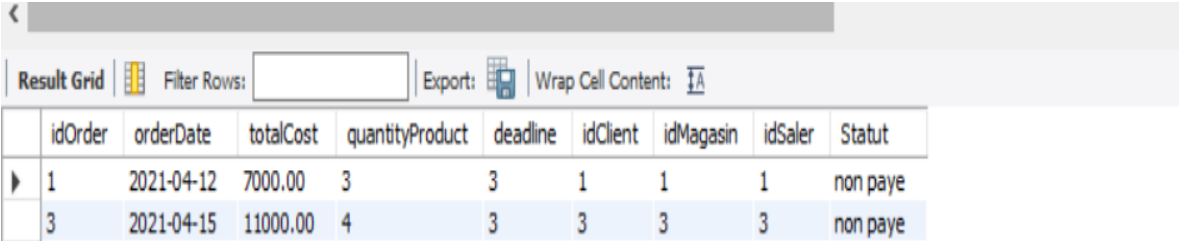
Для нашей базы данных скрипт создания процедуры выглядит следующим образом:

```
DELIMITER //
CREATE PROCEDURE GetOrder()
begin
SELECT * FROM Orderx where orderDate between '2021-04-01'and '2021-
04-30';
end //
call GetOrder();
```

Консольный оператор DELIMITER переопределяет строку, по которой MySQL определяет завершение команды. Задав DELIMITER равным //, мы получаем возможность использовать точку с запятой внутри кода, например создание процедуры.

DELIMITER позволяет передать на сервер разделитель «;» , указанный в теле процедуры, тем самым освобождая mysql от самостоятельной интерпретации данного разделителя.

При вызове указывается имя процедуры.



The screenshot shows a database application window with a 'Result Grid' tab. It contains a table with 10 columns: idOrder, orderDate, totalCost, quantityProduct, deadline, idClient, idMagasin, idSaler, and Statut. There are two rows of data displayed.

	idOrder	orderDate	totalCost	quantityProduct	deadline	idClient	idMagasin	idSaler	Statut
▶	1	2021-04-12	7000.00	3	3	1	1	1	non paye
	3	2021-04-15	11000.00	4	3	3	3	3	non paye

Рисунок 6— Результат выполнения процедура.

### 3.5 Разработка триггеров

Триггер – это подпрограмма, похожая на процедуру БД, автоматически вызываемая СУБД при изменении, удалении или добавлении записи в таблице. К триггерам невозможно обратиться из программы, передать им параметры или получить от них результат. Наиболее часто триггеры применяются для поддержания ссылочной целостности и каскадных операций в БД. Ссылочные спецификации, определяющие каскадные действия при удалении и обновлении и создаваемые при объявлении таблиц, также реализуются через триггеры, однако текст этих триггеров не редактируется.

Назначение триггеров

- Предотвращение изменения

- Журналирование изменения
- Аудит изменений
- Фиксация изменений
- Реализация бизнес-правил.
- Репликация данных
- Повышение производительности

Для нашей базы данных один из триггеров выглядит следующим образом:

```
USE shoesSport;
DELIMITER //
CREATE TRIGGER Price BEFORE INSERT ON Order_has_Sneakers
FOR EACH ROW
BEGIN
  declare var int;
  set var= (select sum(totalPriceOrder) from Order_has_Sneakers where
idOrder=new.idOrder)+
  (select cost from Sneakers where idSneakers= new.idSneakers )*
new.QuantityOrder ;
  set new.totalPriceOrder =(select cost from Sneakers where idSneakers=
new.idSneakers )* new.QuantityOrder;
  update `Orderx`
  set `totalCost`= var where Orderx.idOrder=new.idOrder;
END; //
DELIMITER ;
show triggers;
```

Оператор CREATE TRIGGER позволяет создать триггер. Далее идёт его название, в нашем случае – это Price.

Ключевое слово BEFORE позволяет проверить условие перед выполнением запроса на добавление.



Строка `FOR EACH ROW` указывает на то, что триггер будет запускаться для каждой обрабатываемой командой строки.

Этот триггер позволяет нам вводить правильные значения в столбцы, содержащие общую цену заказанных кроссовки.

## **Заключение**

В ходе работы, мы спроектировали и создали базу данных для магазина спортивной обуви, в которой хранятся данные о покупках и заказах, сделанных в магазине. Также мы выполнили различные запросы, процедуры и триггеры, позволяющие манипулировать сохраненными данными.

Можно сказать, что цель задания была достигнута благодаря успешному выполнению:

- описания предметной области;
- Разработки схем базы данных;
- реализации базы данных.

В заключении можно сказать, что реализованная база данных полностью удовлетворяет требованиям, а также позволит структурировать всю работу магазина спортивной обуви в целом.

## Список литературы

1. Голицына, О. Л. Базы данных / О.Л. Голицына, Н.В. Максимов, И.И. Попов. - М.: Форум, 2015. - 400 с.
2. Илюшечкин, В. М. Основы использования и проектирования баз данных / В.М. Илюшечкин. - М.: Юрайт, Юрайт, 2013. - 224 с.
3. Илюшечкин, В. М. Основы использования и проектирования баз данных. Учебник / В.М. Илюшечкин. - М.: Юрайт, 2014. - 214 с.
4. Костин, А. Е. Организация и обработка структур данных в вычислительных системах. Учебное пособие / А.Е. Костин, В.Ф. Шаньгин. - М.: Высшая школа, 2014. - 248 с.

## Приложение А (обязательное)

### Скрипты создания таблиц базы данных

```
CREATE DATABASE IF NOT EXISTS `shoesSport`;
```

```
USE `shoesSport`;
```

```
CREATE TABLE IF NOT EXISTS `shoesSport`.`Saler` (
```

```
`idSaler` INT NOT NULL AUTO_INCREMENT,
```

```
`lastName` VARCHAR(255) NOT NULL,
```

```
`firstName` VARCHAR(255) NOT NULL,
```

```
`middleName` VARCHAR(255) NOT NULL,
```

```
PRIMARY KEY (`idSaler`))
```

```
ENGINE = InnoDB;
```

```
CREATE TABLE IF NOT EXISTS `shoesSport`.`Clientx` (
```

```
`idClient` INT NOT NULL AUTO_INCREMENT,
```

```
`lastName` VARCHAR(255) NOT NULL,
```

```
`firstName` VARCHAR(255) NOT NULL,
```

```
`middleName` VARCHAR(255) NOT NULL,
```

```
`telNumber` VARCHAR(20) NOT NULL,
```

```
PRIMARY KEY (`idClient`))
```

```
ENGINE = InnoDB;
```

```
CREATE TABLE IF NOT EXISTS `shoesSport`.`Magasin` (
```

```
`idMagasin` INT NOT NULL AUTO_INCREMENT,
```

```
`adress` VARCHAR(255) NOT NULL,
```

```
`MagasinName` VARCHAR(255) NOT NULL,
```

```
PRIMARY KEY (`idMagasin`))
```

```
ENGINE = InnoDB;
```

```
CREATE TABLE IF NOT EXISTS `shoesSport`.`Firme` (  
  `idFirme` INT NOT NULL AUTO_INCREMENT,  
  `firmeName` VARCHAR(255) NOT NULL,  
  PRIMARY KEY (`idFirme`))  
  
ENGINE = InnoDB;  
  
CREATE TABLE IF NOT EXISTS `shoesSport`.`Category` (  
  `idCategory` INT NOT NULL AUTO_INCREMENT,  
  `categoryName` VARCHAR(255) NOT NULL,  
  PRIMARY KEY (`idCategory`))  
  
ENGINE = InnoDB;  
  
CREATE TABLE IF NOT EXISTS `shoesSport`.`Color` (  
  `idColor` INT NOT NULL AUTO_INCREMENT,  
  `colorName` VARCHAR(255) NOT NULL,  
  `quantity` INT NOT NULL,  
  PRIMARY KEY (`idColor`))  
  
ENGINE = InnoDB;  
  
CREATE TABLE IF NOT EXISTS `shoesSport`.`Material` (  
  `idMaterial` INT NOT NULL AUTO_INCREMENT,  
  `materialName` VARCHAR(255) NOT NULL,  
  PRIMARY KEY (`idMaterial`))  
  
ENGINE = InnoDB;  
  
CREATE TABLE IF NOT EXISTS `shoesSport`.`Measure` (  
  `idMeasure` INT NOT NULL AUTO_INCREMENT,  
  `measureValue` INT NOT NULL,
```

```

`quantityM` INT NOT NULL,

PRIMARY KEY (`idMeasure`))

ENGINE = InnoDB;

CREATE TABLE IF NOT EXISTS `shoesSport`.`Model` (

`idModel` INT NOT NULL AUTO_INCREMENT,

`quantity` INT NOT NULL,

`idFirme` INT NOT NULL,

`idCategory` INT NOT NULL,

PRIMARY KEY (`idModel`),

CONSTRAINT `fk_idFirme_1x`

FOREIGN KEY (`idFirme`)

REFERENCES `shoesSport`.`Firme` (`idFirme`)

ON DELETE CASCADE

ON UPDATE CASCADE,

CONSTRAINT `fk_idCategory_1x` FOREIGN KEY (`idCategory`)

REFERENCES `shoesSport`.`Category` (`idCategory`)

ON DELETE CASCADE

ON UPDATE CASCADE)

ENGINE = InnoDB;

CREATE TABLE IF NOT EXISTS `shoesSport`.`Model_has_Material` (

`idModel` INT NOT NULL,

`idMaterial` INT NOT NULL,

PRIMARY KEY (`idModel`, `idMaterial`),

CONSTRAINT `fk_idModel_1x`

```



```

FOREIGN KEY (`idModel`)

REFERENCES `shoesSport`.`Model` (`idModel`)

ON DELETE CASCADE

ON UPDATE CASCADE,

CONSTRAINT `fk_idMaterial_1x` FOREIGN KEY (`idMaterial`)

REFERENCES `shoesSport`.`Material` (`idMaterial`)

ON DELETE CASCADE

ON UPDATE CASCADE)

ENGINE = InnoDB;

CREATE TABLE IF NOT EXISTS `shoesSport`.`Sneakers` (

`idSneakers` INT NOT NULL AUTO_INCREMENT,

`sneakersName` VARCHAR (255) NOT NULL,

`cost` DECIMAL(10,2) NOT NULL,

`sneakQuantity` INT NOT NULL,

`idMeasure` INT NOT NULL,

`idColor` INT NOT NULL,

`idModel` INT NOT NULL,

PRIMARY KEY (`idSneakers`),

CONSTRAINT `fk_idModel_2x`

FOREIGN KEY (`idModel`)

REFERENCES `shoesSport`.`Model` (`idModel`)

ON DELETE CASCADE

ON UPDATE CASCADE,

CONSTRAINT `fk_idMeasure_1x`

FOREIGN KEY (`idMeasure`)

```

```

REFERENCES `shoesSport`.`Measure` (`idMeasure`)

ON DELETE CASCADE

ON UPDATE CASCADE,

CONSTRAINT `fk_idColor_1x` FOREIGN KEY (`idColor`)

REFERENCES `shoesSport`.`Color` (`idColor`)

ON DELETE CASCADE

ON UPDATE CASCADE)

ENGINE = InnoDB;

CREATE TABLE IF NOT EXISTS `shoesSport`.`Sale` (

`idSale` INT NOT NULL AUTO_INCREMENT,

`dateS` DATE,

`totalPrice` DECIMAL(10,2) NOT NULL,

`guarantee` INT NOT NULL,

`product_quantity` INT NOT NULL,

`idMagasin` INT NOT NULL,

`idSaler` INT NOT NULL,

PRIMARY KEY (`idSale`),

CONSTRAINT `fk_idMagasin_1x`

FOREIGN KEY (`idMagasin`)

REFERENCES `shoesSport`.`Magasin` (`idMagasin`)

ON DELETE CASCADE

ON UPDATE CASCADE,

CONSTRAINT `fk_idSaler_1x`

FOREIGN KEY (`idSaler`)

REFERENCES `shoesSport`.`Saler` (`idSaler`)

```

ON DELETE CASCADE

ON UPDATE CASCADE)

ENGINE = InnoDB;

CREATE TABLE IF NOT EXISTS `shoesSport`.`Sale\_has\_Sneakers` (

`idSale` INT NOT NULL,

`idSneakers` INT NOT NULL,

`QuantitySale` INT NOT NULL,

`totalPrice` DECIMAL(10,2) NOT NULL,

PRIMARY KEY (`idSale`, `idSneakers`),

CONSTRAINT `fk\_idSale\_1x`

FOREIGN KEY (`idSale`)

REFERENCES `shoesSport`.`Sale` (`idSale`)

ON DELETE CASCADE

ON UPDATE CASCADE,

CONSTRAINT `fk\_idSneakers\_1x`

FOREIGN KEY (`idSneakers`)

REFERENCES `shoesSport`.`Sneakers` (`idSneakers`)

ON DELETE CASCADE

ON UPDATE CASCADE)

ENGINE = InnoDB;

CREATE TABLE IF NOT EXISTS `shoesSport`.`Orderx` (

`idOrder` INT NOT NULL AUTO\_INCREMENT,

`orderDate` DATE,

`totalCost` DECIMAL(10,2) NOT NULL,

`quantityProduct` INT NOT NULL,

```

`deadline` INT NOT NULL,

`idClient` INT NOT NULL,

`idMagasin` INT NOT NULL,

`idSaler` INT NOT NULL,

`Statut` VARCHAR(255) NOT NULL,

PRIMARY KEY (`idOrder`),

CONSTRAINT `fk_idMagasin_2x`

FOREIGN KEY (`idMagasin`)

REFERENCES `shoesSport`.`Magasin` (`idMagasin`)

ON DELETE CASCADE

ON UPDATE CASCADE,

CONSTRAINT `fk_idSaler_2x`

FOREIGN KEY (`idSaler`)

REFERENCES `shoesSport`.`Saler` (`idSaler`)

ON DELETE CASCADE

ON UPDATE CASCADE,

CONSTRAINT `fk_idClient_1x`

FOREIGN KEY (`idClient`)

REFERENCES `shoesSport`.`Clientx` (`idClient`)

ON DELETE CASCADE

ON UPDATE CASCADE)

ENGINE = InnoDB;

CREATE TABLE IF NOT EXISTS `shoesSport`.`Order_has_Sneakers` (

`idOrder` INT NOT NULL,

`idSneakers` INT NOT NULL,

```

```
`QuantityOrder` INT NOT NULL,  
  
`totalPriceOrder` DECIMAL(10,2) NOT NULL,  
  
PRIMARY KEY (`idOrder`, `idSneakers`),  
  
CONSTRAINT `fk_idOrder_1x`  
  
FOREIGN KEY (`idOrder`)  
  
REFERENCES `shoesSport`.`Orderx` (`idOrder`)  
  
ON DELETE CASCADE  
  
ON UPDATE CASCADE,  
  
CONSTRAINT `fk_idSneakers_2x`  
  
FOREIGN KEY (`idSneakers`)  
  
REFERENCES `shoesSport`.`Sneakers` (`idSneakers`)  
  
ON DELETE CASCADE  
  
ON UPDATE CASCADE)  
  
ENGINE = InnoDB;
```

## **Приложение Б (обязательное)**

### **Скрипты заполнения базы данных**

```
use shoesSport;
```

```
show columns from Saler;
```

```
insert into Saler(idSaler, lastName, firstName,middleName)values
```

```
(1, 'Joseph','Wen','Josa'),
```

```
(2, 'Josette','Macelus','Shwenny'),
```

```
(3, 'Josepha','May','Sagnon');
```

```
use shoesSport;
```

```
show columns from Clientx;
```

```
insert into Clientx(idClient, lastName, firstName, middleName, telNumber)values
```

```
(1, 'Moise','Registe','Olem','892566221574'),
```

```
(2, 'Abraham','Jules','Andre','89532664521'),
```

```
(3, 'Hebert','Dean','Phillip','87854565218');
```

```
use shoesSport;
```

```
show columns from Magasin;
```

```
insert into Magasin(idMagasin, adress, MagasinName)values
```

```
(1, 'Baraderes', 'WenSport'),
```

```
(2, 'Cap-haitien','WenSport'),
```

```
(3, 'Jacmel','WenSport');
```

```
use shoesSport;
```

```
show columns from Firme;
```

```
insert into Firme(idFirme, firmeName)values
```

```
(1, 'Adidas'),
```

```
(2, 'Nike'),
```

```
(3, 'Pumas');
```

```
use shoesSport;
```

```
show columns from Category;
```

```
insert into Category(idCategory, categoryName)values
```

```
(1, 'football'),
```

```
(2, 'running'),
```

```
(3, 'basket-ball');
```

```
use shoesSport;
```

```
show columns from Color;
```

```
insert into Color(idColor, colorName, quantity)values
```

```
(1, 'Rouge', 50 ),
```

```
(2, 'Noir', 50),
```

```
(3, 'Jaune', 50);
```

```
use shoesSport;
```

```
show columns from Material;
```

```
insert into Material(idMaterial, materialName)values
```

```
(1, 'textile'),
```

```
(2, 'Cuir'),
```

```
(3, 'Peau');
```

```
use shoesSport;
```

```
show columns from Measure;
```

```
insert into Measure(idMeasure, measureValue, quantityM)values
```

```
(1, 37, 50 ),
```

```
(2, 39, 50),
```

```
(3, 43, 50);
```

```
use shoesSport;
```

```
show columns from Model;
```

```
insert into Model(idModel,quantity,idFirme, idCategory)values
```

```
(1, 50, 1, 1 ),
```

```
(2, 50, 2, 2),
```

```
(3, 50, 3, 3);
```

```
use shoesSport;
```

```
show columns from Model_has_Material;
```

```
insert into Model_has_Material(idModel,idMaterial)values
```

```
(1, 2),
```

```
(2, 2),
```

```
(3, 3);
```

```
use shoesSport;
```

```
show columns from Sneakers;
```



```
insert into Sneakers(idSneakers,sneakersName,cost,sneakquantity,idMeasure, idColor,
idModel)values
```

```
(1, 'Speedy',3000, 100,1, 1,1 ),
(2, 'Mercurial',3500, 100,2, 2,2),
(3, 'Predator',4500, 100,3, 3,3),
(4, 'Danilo',3500, 100,2, 3,2),
(5, 'Sternilo',6000, 100,3, 1,2),
(6, 'Moizo',6000, 150,3, 2,2);
```

```
use shoesSport;
```

```
show columns from Sale;
```

```
insert into Sale(idSale,dateS,totalPrice,guarantee, product_quantity, idMagasin,
idSaler)values
```

```
(1,'2021-04-12', 7000, 1,3, 1,1 ),
(2,'2021-09-21', 9500, 1,2, 2,2),
(3,'2021-11-15', 11000, 1,4, 3,3);
```

```
use shoesSport;
```

```
show columns from Sale_has_Sneakers;
```

```
insert into Sale_has_Sneakers(idSale,idSneakers,QuantitySale, totalPrice)values
```

```
(1,1,2,5000),
(2, 2, 3, 7000),
(3, 3, 1, 5000);
```

```
use shoesSport;
```

```
show columns from Orderx;
```

```
insert into Orderx(idOrder,orderDate,totalCost, quantityProduct, deadline,idClient,  
idMagasin, idSaler, statut)values
```

```
(1,'2021-04-12', 7000, 3, 3,1,1,1, 'non paye' ),
```

```
(2,'2021-09-21', 9500, 2, 3,2,2,2,'non paye'),
```

```
(3,'2021-04-15', 11000, 4, 3,3,3,3, 'non paye');
```

```
use shoesSport;
```

```
show columns from Order_has_Sneakers;
```

```
insert          into          Order_has_Sneakers(idOrder,idSneakers,QuantityOrder,  
totalPriceOrder)values
```

```
(1,1,2,5000),
```

```
(2, 2, 3, 7000),
```

```
(3, 3, 1, 5000);
```

## Приложение В (обязательное)

### Реализация запросов к базе данных

```

use shoesSport;

select    Sneakers.sneakquantity,    Sneakers.sneakersName,    Firme.firmeName,
Color.colorName

from (((Sneakers

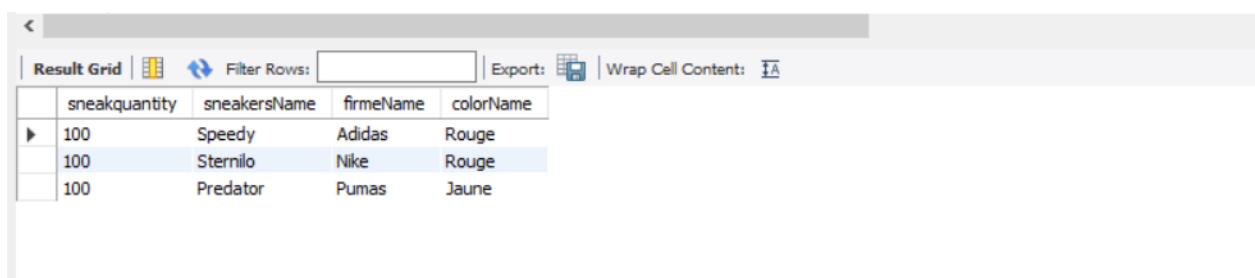
inner join Color on Sneakers.idColor = Color.idColor)

inner join Model on Sneakers.idModel = Model.idModel)

inner join Firme on Model.idFirme= Firme.idFirme)

group by Sneakers.idModel;

```



	sneakquantity	sneakersName	firmeName	colorName
▶	100	Speedy	Adidas	Rouge
	100	Sternilo	Nike	Rouge
	100	Predator	Pumas	Jaune

Рисунок 7 – Результат запроса 1

```

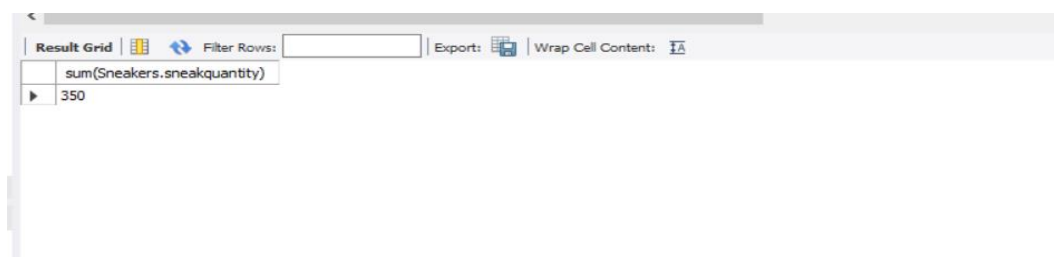
use shoesSport;

select sum(Sneakers.sneakquantity)

from Sneakers

where Sneakers.idMeasure=(select Measure.idMeasure from Measure where
measureValue=43);

```



	sum(Sneakers.sneakquantity)
▶	350

Рисунок 8 – Результат запроса 2

## Приложение Г (обязательное)

### Реализация триггеров и процедур

```
DELIMITER //
```

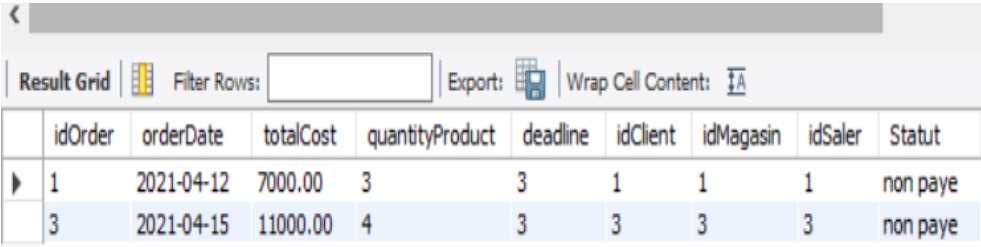
```
CREATE PROCEDURE GetOrder()
```

```
begin
```

```
SELECT * FROM Orderx where orderDate between '2021-04-01'and '2021-04-30';
```

```
end //
```

```
call GetOrder();
```



The screenshot shows a database interface with a 'Result Grid' tab. It displays two rows of data from a query. The columns are: idOrder, orderDate, totalCost, quantityProduct, deadline, idClient, idMagasin, idSaler, and Statut. The first row has values: 1, 2021-04-12, 7000.00, 3, 3, 1, 1, 1, non paye. The second row has values: 3, 2021-04-15, 11000.00, 4, 3, 3, 3, 3, non paye.

	idOrder	orderDate	totalCost	quantityProduct	deadline	idClient	idMagasin	idSaler	Statut
▶	1	2021-04-12	7000.00	3	3	1	1	1	non paye
	3	2021-04-15	11000.00	4	3	3	3	3	non paye

Рисунок 9— Результат процедура

```
CREATE PROCEDURE GetClient(IN firstN varchar(255))
```

```
begin
```

```
SELECT * FROM Clientx where firstName= firstN ;
```

```
end //
```

```
call GetClient('Jules');
```

```
USE shoesSport;
```

```
DELIMITER //
```

```
CREATE TRIGGER Price BEFORE INSERT ON Order_has_Sneakers
```

```
FOR EACH ROW
```

```
BEGIN
```

```
declare var int;
```

```

        set var= (select sum(totalPriceOrder) from Order_has_Sneakers where
idOrder=new.idOrder)+

        (select cost from Sneakers where idSneakers= new.idSneakers )* new.QuantityOrder ;

        set new.totalPriceOrder =(select cost from Sneakers where idSneakers= new.idSneakers
)* new.QuantityOrder;

        update `Orderx`

        set `totalCost`= var where Orderx.idOrder=new.idOrder;

END; //

DELIMITER ;

show triggers;

USE shoesSport;

DELIMITER //

CREATE TRIGGER PriceVente BEFORE INSERT ON Sale_has_Sneakers

FOR EACH ROW

BEGIN

declare var int;

set var= (select sum(totalPrice) from Sale_has_Sneakers where idSale=new.idSale)+

(select cost from Sneakers where idSneakers= new.idSneakers )* new.QuantitySale ;

set new.totalPrice =(select cost from Sneakers where idSneakers= new.idSneakers )*
new.QuantitySale;

        update `Sale`

        set `totalPrice`= var where Sale.idSale=new.idSale;

END; //

DELIMITER ;

```

```
USE shoesSport;

CREATE TRIGGER add_Sneakers BEFORE INSERT ON Sneakers

FOR EACH ROW

BEGIN

DECLARE msg VARCHAR(255);

IF (new.cost<=0) THEN

SIGNAL SQLSTATE '45000';

SET msg = 'Цена не может быть меньше или равна 0!';

END IF;

END//
```