Федеральное государственное бюджетное образовательное учреждение
высшего профессионального образования
«**Московский Государственный Технический Университет
имени Н. Э. Баумана**»

Курсовая работа

по дисциплине: «Базы данных»

на тему:

«База данных интернет-магазина»

Выполнила:
студентка 3 курса,
группы ИУ 9-62
Козлова А.А.

Руководитель:
Вишняков И.Э.

Москва 2017

Содержание

Введение.

Развитие методов и средств обработки данных в информационных системах привело к появлению концепции базы данных. Основной особенностью этой концепции является представление, как самих данных, так и их описания в запоминающей среде ЭВМ.

В настоящее время базы данных составляют основу компьютерного обеспечения информационных процессов, входящих практически во все сферы человеческой деятельности. Различные базы данных используются для систематизации и хранения большого количества однотипных документов и быстрого доступа к ним.

В современном мире работа с базами данных является важнейшим навыком в работе с компьютером, а специалисты данной области становятся всё более востребованными.

Представление, хранение и обработка информации с помощью баз данных используется во многих предметных областях, в том числе и в сфере торговли. Торговый бизнес считается одним из самых популярных видов предпринимательской деятельности, и с каждый годом это направление развивается все больше и больше. В настоящее время стремительно развивается и набирает популярность торговля в глобальной сети интернет. Для этих целей создается множество интернет-магазинов, которые позволяют заказывать товары из магазинов, расположенных в самых различных местах нашей планеты.

Целью данного курсового проекта является создание базы данных интернет-магазина и пользовательского приложения для управления этой базой.

1. Обзор предметной области и проектирование ER-модели базы данных

1. 1 Обзор предметной области.

В современном мире все большую популярность приобретают покупки с помощью интернета, что способствует значительному развитию и укреплению в торговой сфере интернет-магазинов. На данный момент существует огромное количество различных интернет-магазинов, и абсолютно каждый из них имеет свою базу данных, в которой хранится информация, необходимая для совершения торговых сделок.

Основной информацией, которая содержится в базах данных для интернет-магазинов, являются сведения о товарах, покупателях и поставщиках. Это объясняется тем, что для успешного функционирования интернет-магазина его работникам и владельцам необходимо систематизировано хранить список продаваемых товаров и иметь быстрый доступ к информации о любом продукте из данного списка. Кроме того, чтобы отправлять пользователю интернет-магазина приобретенные им товары, в базе должны быть сведения об адресе покупателя, а также другая не менее важная информация для успешного взаимодействия с ним, например, личный идентификационный номер, имя, фамилия и т. д. Также следует заметить, что у каждого товара есть свои поставщики и для наилучшей коммуникации с ними важно иметь в базе данных сведения о самих фирмах-поставщиках и о поставках, которые они совершают.

1. 2 Разработка ER-модели базы данных интернет-магазина.

Первым шагом для создания ER-модели были выбраны основные сущности для базы данных интернет-магазина.

Самой первой была создана сущность «товар», которая является одной из ключевых сущностей для базы данных любого магазина. Эта сущность крайне важна как для покупателя, которому может быть предоставлена возможность посмотреть какие товары он потенциально может приобрести в данном

магазине и за какую цену, так и для продавца. Далее был выбран идентификатор сущности. Первоначально в его роли выступало название товара, но потом стало понятно, что это не совсем корректно, т. к. продаваемый продукт может иметь одно название, но, например, разных поставщиков, и было принято решение сделать числовой идентификатор, который представляет собой штрихкод товара. Затем были добавлены атрибуты для данной сущности, а именно:

- название товара
- категория товара (например, овощи, фрукты, молочные продукты и т.д.)
- цена за единицу товара
- количество в наличии
- эквивалент (единица измерения товара, например, килограммы)

После этого была создана не менее важная для предметной области проектируемой базы данных сущность «пользователь», которая содержит в себе сведения о потенциальных покупателях. Для ее идентификатора было выбрано уникальное числовое значение, которое должно присваиваться каждому пользователю, прошедшему регистрацию в интернет-магазине. Для данной сущности были созданы следующие атрибуты:

- логин
- пароль
- e-mail
- ник

Причем любой из этих атрибутов имеет уникальное значение для каждого пользователя.

Логин и пароль используется для входа пользователя в личный кабинет на сайте интернет-магазина. Ник — это имя, выбранное пользователем, которое будет отображаться при комментировании или написании отзывов о товарах. E-mail нужен для оповещения пользователей о доставке товара, а также для рассылки информации о проходящих акциях и скидках.

Затем была добавлена сущность под названием «поставщик», в которой должна хранится информация о фирмах, поставляющих товары в интернет-магазин. Было принято решение идентификатором данной сущности сделать название фирмы. В качестве атрибутов для сущности «поставщик» были выбраны следующие данные:

- телефон
- официальный сайт

Так как в интернет-магазине предполагается наличие онлайн продавцов-консультантов, которые помогают покупателям в случае спорных ситуаций и предоставляют интересующую пользователей информацию о продаваемом продукте, возникла необходимость добавить в ER-модель сущность «продавец-консультант». Идентификатором данной сущности стал уникальный номер, который закрепляется за каждым продавцом, зарегистрированным на сайте. Также были подобраны атрибуты для сущности «продавец-консультант»:

- имя
- фамилия
- отчество
- ник
- страна

Ник — псевдоним, который будет виден пользователям, вступающим в диалог с продавцом. Информация о том, в какой стране проживает консультант, необходима для интернет-магазинов международного уровня, например, таких как Aliexpress.

Итак, после того как были созданы все основные сущности было необходимо добавить связи между ними, а также вспомогательные идентификационно-зависимые сущности.

Следующим шагом был создан способ взаимодействия между сущностями «товар» и «поставщик». По причине того, что фирмы-поставщики могут поставлять множество продуктов, а у товаров одного вида может быть несколько поставщиков, была создана вспомогательная идентификационно-зависимая сущность «поставки», в которой содержится информация только об определенном товаре, поставляемом некоторым поставщиком. Идентификатор новой сущности представляет собой пару, состоящую из значений идентификаторов сущностей «товар» и «поставщик». В качестве атрибутов созданной сущности выступают цена за единицу поставляемого товара одного вида и количество этого товара. Связь между сущностями «поставщик» и «поставки» была определенна, как «один-ко-многим», так как у фирмы-поставщика может быть большое количество различных поставок, а у каждой поставки может быть только один конкретный поставщик. Такой же тип связи был установлен между сущностями «товар» и «поставки». Это объясняется тем, что в одной поставке может быть только определенный товар, а сам продукт может входить в несколько поставок.

Затем были установлены отношения между сущностями «продавец-консультант» и «товар». В данном случае была выбрана связь «многие-ко-многим», так как за продавцом-консультантом может быть закреплено большое количество товаров, а у одного продукта может быть несколько продавцов-консультантов.

Для хранения информации о покупке, совершенной пользователем, была создана сущность «заказ». По причине того, что каждый заказ должен быть закреплен за определенным покупателем, новая сущность была определена, как идентификационно-зависимая от сущности «пользователь». Идентификатор сущности «заказ» состоит из номера заказа и идентификатора пользователя интернет-магазина, который сделал данный заказ. Атрибутами для созданной сущности служат дата совершения заказа, его статус и общая сумма, которую покупателей должен заплатить за все заказанные товары. Так как пользователей может сделать несколько заказов, а каждый заказ закреплен за одним единственным пользователем, был сделан вывод, что связь между сущностями «пользователь» и «заказ» имеет тип «один-ко-многим».

Далее потребовалось определить отношение между сущностями «заказ» и «товар».

Чтобы в заказе могло содержаться несколько товаров, была создана идентификационно-зависимая сущность «товар в заказе», в которой отображается информация об одном определенном товаре в некотором заказе. В качестве идентификатора данной сущности выступают объединение трех значений, а именно:

1. идентификатор пользователя, который совершил покупку
2. номер заказа
3. идентификатор заказываемого товара

Атрибутами добавленной сущности являются цена за единицу определенного товара и его количество.

Затем нужно было определить тип связи между сущностями «заказ» и « товар в заказе». Так как товаров в одном заказе может быть много, а каждый товар из заказа, благодаря тому, что в идентификаторе содержится номер заказа, может

принадлежать только какому-то конкретному заказу, был сделан вывод, что типом связи между данными сущностями является связь «один-ко-многим».

Далее также была добавлена связь «один-ко-многим» между сущностями «товар» и «товар в заказе».

Почти во всех популярных интернет-магазинах все товары имеют свой рейтинг и каждый зарегистрированный пользователь может оценить продукт, поставив ему соответствующую оценку, поэтому была создана ещё одна идентификационно-зависимая сущность по названием «рейтинг». Идентификатором данной сущности состоит из идентификатора пользователя, оценившего продукт, и идентификатора товара, который был оценен этим конкретным пользователем. Атрибутом сущности «рейтинг» является количество звезд, поставленный тем или иным покупателем тому или иному товару.

Затем для того, чтобы можно было хранить адрес пользователя в наиболее удобном формате, была создана сущность «адрес», которая является дочерней для сущности «пользователь». Идентификатором созданной сущности служит идентификатор пользователя, который проживает по данному адресу. Сущность «адрес» имеет следующие атрибуты:

- квартира
- дом
- улица
- страна
- почтовый индекс

Связь между сущностями «пользователь» и «адрес» была определена, как «один-ко-многим», так как предполагается, что пользователь может указать

несколько адресов, на которые может осуществляться доставка товара, а тот или иной адрес может быть закреплен только за одним пользователем .

В итоге, получилась диаграмма построенной ER-модели, представленная на рисунке №1.

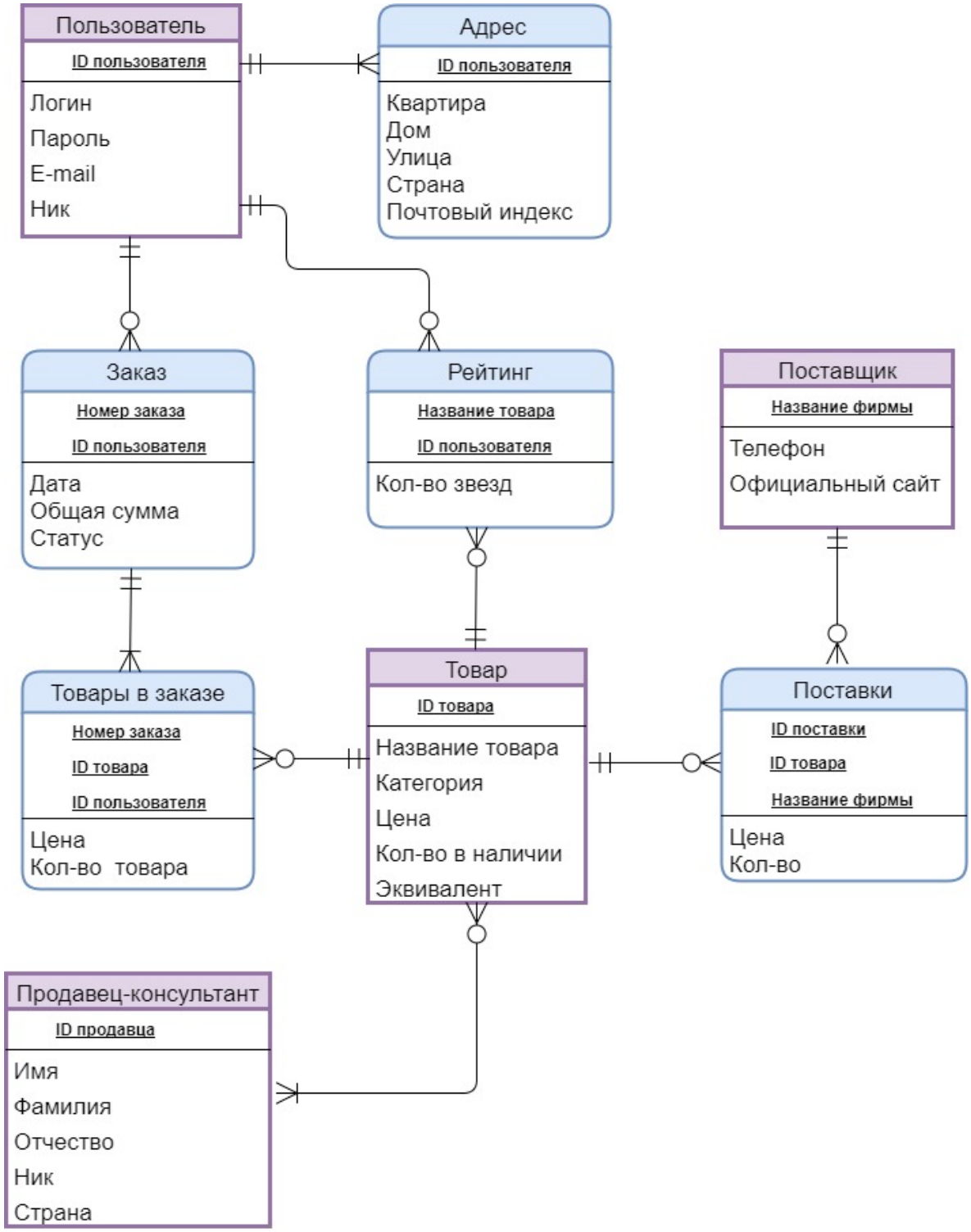

Рисунок №1. ER-модель базы данных.

2. Создание базы данных и проектирование пользовательского приложения.

2.1. Преобразование ER-модели базы данных в реляционную модель.

Следующим шагом в проектировании баз данных является преобразование ER-модели в конкретную схему базы данных на основе выбранной модели данных.

Реляционные базы данных получили очень широкое распространение и считаются самыми популярными и удобными в использовании, поэтому в курсовой проекте будет использоваться именно этот тип базы данных.

При преобразовании ER-модели базы данных в реляционную модель все сущности, которые были определены, необходимо представить в виде отношений, т. е. двумерных таблиц. Для этого нужно определить тип данных, которые будут находиться в столбце таблицы, установить могут ли в нем присутствовать значения null, а также выделить ключи.

Было определено, что значение null может быть в отношении «пользователь» в столбце «E-mail», так как указание электронного почтового адреса является необязательным для покупателя. Также в таблице «товар» в столбце «категория» может не быть указанно какой-либо информации, потому что разделение товара на категории является произвольным и допускается, что определенный товар может не попадать ни в одну из них. Во всех остальных столбцах этих и других отношений данные обязательно должны быть указаны.

Первичным ключом для таблицы в реляционной модели будет служить идентификатор соответствующей сущности в ER-модели. Также для связей «один-ко-многим» между таблицами необходимо добавить в дочерние таблицы внешние ключи, которые представляют собой идентификаторы в родительской таблице.

Выбор типа данных и определение ключей для каждого отношения отражено в таблицах 1-9.

Таблица 1. Определение типа данных и ключей для отношения «пользователь»

Пользователь		
Имя столбца	**Тип**	**Ключ**
ID пользователя	int	Primary
Логин	char(100)	Alternative
Пароль	char(100)	Alternative
E-mail	char(100)	NO

Таблица 2. Определение типа данных и ключей для отношения «адрес»

Адрес		
Имя столбца	**Тип**	**Ключ**
ID пользователя	int	Primary and Foreign
Дом	char(5)	NO
Улица	char(100)	NO
Страна	char(100)	NO
Почтовый индекс	int	Alternative

Таблица 3. Определение типа данных и ключей для отношения «заказ»

Заказ		
Имя столбца	**Тип**	**Ключ**
Номер заказа	int	Primary
ID пользователя	int	Primary and Foreign
Дата	timestamp	No
Общая сумма	money	No
Статус	char(100)	No

Таблица 4. Определение типа данных и ключей для отношения «рейтинг»

Рейтинг		
Имя столбца	**Тип**	**Ключ**
Название товара	char(100)	Primary and Foreign
ID пользователя	int	Primary and Foreign
Количество звезд	int	No

Таблица 5. Определение типа данных и ключей для отношения «поставщик»

Поставщик		
Имя столбца	**Тип**	**Ключ**
Название фирмы	char(100)	Primary
Телефон	char(100)	Alternative
Официальный сайт	char(100)	Alternative

Таблица 6. Определение типа данных и ключей для отношения «товар»

Товар		
Имя столбца	**Тип**	**Ключ**
Название товара	char(100)	Primary
Категория	char(100)	NO
Цена	money	NO
Количество в наличии	int	NO
Эквивалент	char(20)	NO

Таблица 7. Определение типа данных и ключей для отношения «товар в заказе»

Товар в заказе		
Имя столбца	**Тип**	**Ключ**
Название товара	char(100)	Primary and Foreign
Номер заказа	int	Primary and Foreign

Цена	money	NO
Количество товара	int	NO

Таблица 8. Определение типа данных и ключей для отношения «поставки»

Поставки		
Имя столбца	**Тип**	**Ключ**
Номер поставки	int	Primary
Название товара	char(100)	Primary and Foreign
Название фирмы	char(100)	Primary and Foreign
Цена	money	NO
Количество товара	int	NO

Таблица 9. Определение типа данных и ключей для отношения «продавец-консультант»

Продавец-консультант		
Имя столбца	**Тип**	**Ключ**
ID продавца	char(100)	Primary
Имя	char(100)	NO
Фамилия	money	NO
Ник	int	Alternative
страна	char(100)	NO

Следующим шагом стал выбор действий для поддержания ссылочной целостности таблиц.

Первой была рассмотрена таблица «поставщик» и связанная с ней дочерняя таблица «поставки», внешний ключ которой является также первичным. При вставке первичного ключа в таблицу «поставщик» и при удалении первичного ключа из отношения «поставки» не нужно производить никаких дополнительных действий. При вставке новой записи в таблицу «поставки»

необходимо осуществить подбор новой родительской записи из таблицы «поставщик», потому что внешний ключ обязательно должен иметь какое-либо значение, которое в данном случае должно совпадать с одним из первичных ключей в таблице «поставщик». Обновление первичного ключа в таблице «поставки» допустимо только, если значение ключа соответствует некоторому первичному ключу в родительской таблице. При изменении первичного ключа в таблице «поставщик» необходимо применить данные изменения ко всем строкам в таблице «поставки», где этот ключ является внешним, следовательно нужно использовать каскадное изменение. По тем же соображениям при удалении строки из таблицы «поставщик» следует применять каскадное удаление.

После этого было выявлено, что для таблиц «товар» - «поставки», «товар» - «товар продавца», «пользователь»-«рейтинг» и «пользователь» - «заказ» выполняются аналогичные действия.

Для таблиц «товар» - «рейтинг» нужно выполнять точно такие же действия, кроме случая, когда происходит удаление из дочерней записи, по причине того, что нельзя удалить рейтинг у какого-либо товара.

Действия для остальных пар таблиц похожи, но имеют свою специфику.

Затем были рассмотрены таблицы «заказ», которая является родительской, и «товары в заказе», которая является дочерней. При вставке в таблицу «заказ» необходимо сделать подбор новой дочерней записи, так как заказ не может быть пустым и не содержать товаров, приобретенных пользователем. Обновление первичного ключа, в родительской и дочерней таблице запрещено, т. к. каждый номер заказа уже закреплен за определенным пользователем. При удалении строк из таблицы «заказ» целесообразно применять каскадное удаление. При вставке в таблицу «товар в заказе» новой строки нужно осуществлять подбор новой родительской записи. Удаление из дочерней таблицы возможно только в том случае, если есть другие дочерние записи у соответствующей родительской,

дополнительные действия не нужны, если запись является единственной, то требуется удалить и родительскую запись.

Следующим шагом были рассмотрены таблицы «продавец-консультант», которая является родительской, и «товары продавца». При вставке в таблицу «продавец-консультант» необходимо выполнять подбор новой дочерней записи , потому что за каждым продавцом должен быть закреплен хотя бы товар. Также в родительской таблице необходимо реализовать каскадное удаление и обновление. При вставке в дочернюю таблицу нужно совершать подбор новой записи из таблицы «продавец-консультант».Обновление первичного ключа в таблице «товары продавца» допустимо, если новое значение ключа соответствует некоторому первичному в родительской таблице. Удаление из дочерней таблицы можно осуществлять только, если у родительской таблицы есть другие дочерние записи.

Последними были рассмотрены таблицы «пользователь» и «адрес», которая является дочерней. При вставке в родительскую таблицу нужно совершать подбор новой записи в таблице «адрес», так как при регистрации на сайте интернет-магазина каждый пользователь обязан указать хотя бы один адрес, по которому можно будет отправить посылку с заказом. Удаление и обновление в таблице «пользователь» было принято решение сделать каскадным. При вставке в дочернюю таблицу необходимо подобрать новую запись в родительской таблице, чтобы внешний ключ таблицы «адрес» соответствовал одному из первичных в таблице «пользователь». Удаление из дочерней таблицы или ее изменение запрещено.

Следующим шагом для связи «многие-ко-многим» между таблицами «товар» и «продавец-консультант» была создано промежуточное отношение «товары продавца». Связь между таблицами «товар» и «продавец-консультант» и созданной таблицей определяется, как «один-ко-многим». Первичный ключ

таблицы «товары продавца» состоит из первичных ключей родительских таблиц, которые в свою очередь являются внешними в данном отношении.

Получившаяся реляционная модель отражена на рисунке №2.

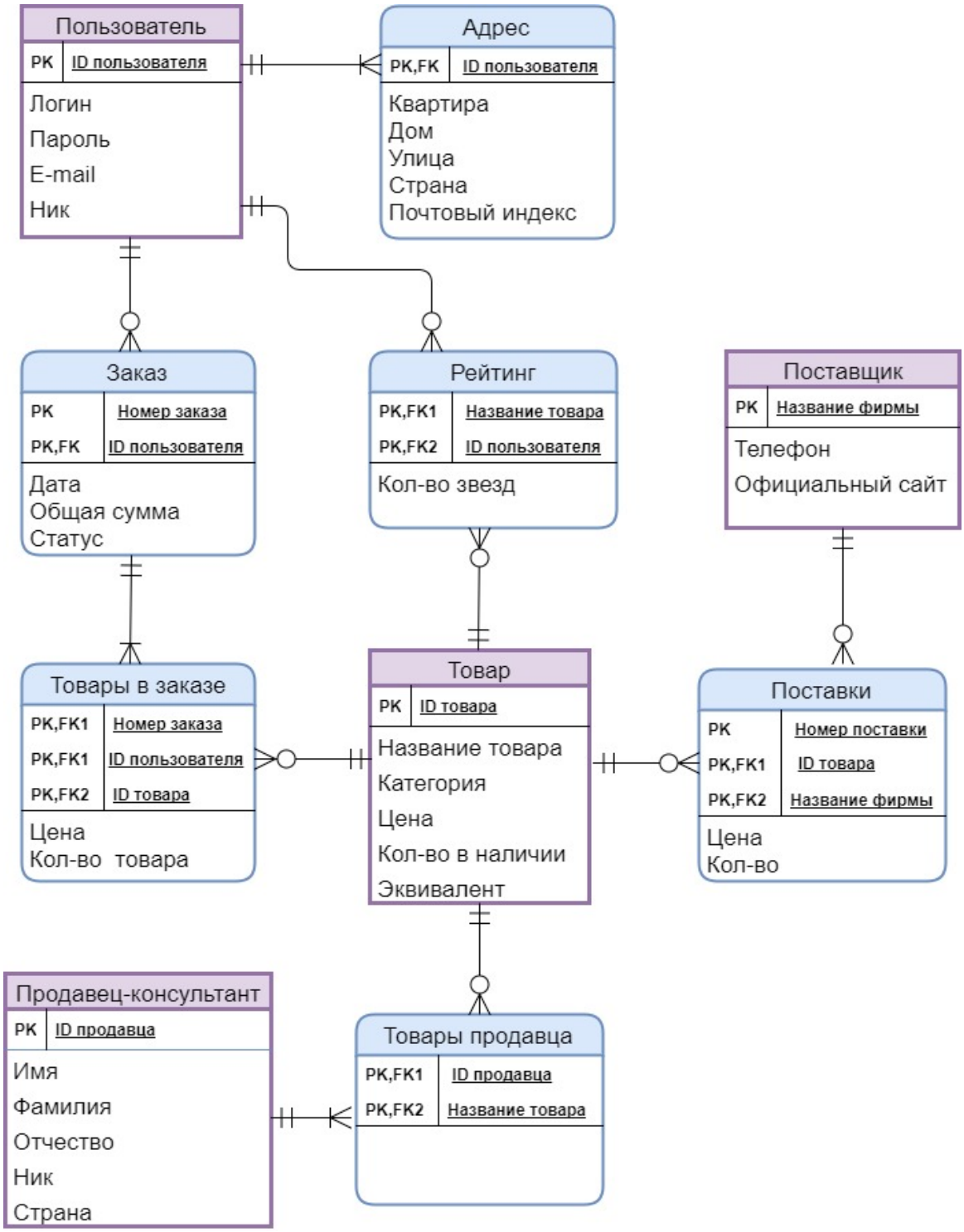

Рисунок №2. Реляционная модель базы данных.

2.2. Выбор СУБД и реализация базы данных

При реализации базы данных была использована самая популярная и функциональная СУБД для PostgreSQL — pgAdminIII.

Для создания базы данных в выбранной СУБД использовался язык запросов pgSQL, который является процедурным расширением языка SQL.

Вначале были созданы таблицы с указанием внешних и первичных ключей для каждой из них. Затем для реализации бизнес-логики базы данных были написаны триггеры.

Для таблицы, в которой хранится список поставок в интернет-магазин, потребовалось создать триггеры на вставку и обновление.

В триггере «insert» после вставки нового значения в таблицу «Поставки» происходит «update» таблицы «Товар», где к «старому» количеству имеющегося товара прибавляется количество поставляемого товара из вставляемой строки. Этот триггер представлен на листинге №1.

```
1 CREATE FUNCTION delivery_insert() RETURNS TRIGGER AS $delivery_insert$
2 BEGIN
3     UPDATE products
4     SET count_product = count_product + NEW.amount
5     WHERE id_product=NEW.fk_id_product;
6 END;
7 $delivery_insert$ LANGUAGE plpgsql;
8
9 CREATE TRIGGER Delivery_insert
10    AFTER INSERT ON Delivery
11    FOR EACH ROW
12. EXECUTE PROCEDURE delivery_insert();
```

Листинг №1. Триггер вставки в таблицу «Поставки»

В триггере «update» после обновления данных в какой-либо строке таблицы «Поставки» проверяется произошли ли изменения в ячейке, отвечающей за количество поставляемого товара, и, если это так, из имеющего количества продукта в таблице «Товар» вычитается «старое» значение обновляемой ячейки и прибавляется «новое».

Также триггеры для вставки, удаления и обновления данных понадобились для таблицы «Товар в заказе».

Для триггера «insert» этой таблицы был выбран тип BEFORE, так как необходимо вначале проверить есть ли запрашиваемое количество товара на складе магазина (информация об этом хранится в столбце «Кол-во в наличии» таблицы «Товар»). Из-за того, что триггер отрабатывает до непосредственной вставки данных в таблицу «Товар в заказе», нужно проверять существуют ли идентификаторы товара и пользователя, указанные во вставляемой строке, в соответствующих таблицах «Товар» и «Пользователь» и, если таковых нет, то выдавать RAISE EXCEPTION с информацией об ошибке. После этого необходимо проверить есть ли на складе нужное количество товара, если нет, то сообщить об этом с помощью RAISE EXCEPTION. Если все предыдущие условия выполняются, то происходит обновление таблицы «Товар», а именно в соответствии с количеством товара, которое указано в заказе, уменьшается его число на складе магазина. Затем проверяется существует ли уже идентификатор заказа, который указан во вставляемой строке, в таблице заказов, если нет, то формируется сумма заказа и выполняется вставка новой записи в таблицу «Заказ». Если же вставка товара происходит в уже существующий заказ, то выполняется обновление итоговой суммы этого заказа в зависимости от количества и цены товара. Кроме того, с помощью данного триггера осуществляется возможность вставки в таблицу строки, в которой не указана стоимость товара, потому что в этом случае будет вставлена и использована при подсчете итоговой суммы цена, которая указана в таблице «Товар» для вставляемого продукта. Код триггера приведен в листинге №2.

```
CREATE FUNCTION product_in_order_insert() RETURNS TRIGGER AS $product_in_order_inser
t$
DECLARE
  old_count           INTEGER;
  sum_price           INTEGER;
BEGIN
  IF (NOT EXISTS (SELECT id_product FROM products
                  WHERE id_product = NEW.fk_id_product)) THEN
       RAISE EXCEPTION 'Товара с данным идентификатором не существует';
  ELSIF (NOT EXISTS (SELECT id_user FROM users
                     WHERE id_user = NEW.fk_id_user)) THEN
       RAISE EXCEPTION 'Пользователя с данным идентификатором не
существует';
  END IF;
  old_count = (SELECT count_product FROM products
               WHERE id_product = NEW.fk_id_product);
  IF (old_count >= NEW.amount) THEN
    UPDATE products
    SET count_product = count_product - NEW.amount
    WHERE id_product= NEW.fk_id_product;
    IF (NEW.price IS NULL) THEN
      NEW.price = (SELECT price FROM products
                   WHERE id_product = NEW.fk_id_product);
```

```
    END IF;
    IF (NOT EXISTS (SELECT order_number FROM orders
                    WHERE order_number = NEW.fk_order_number)) THEN
      sum_price = NEW.amount * NEW.price;
      INSERT INTO orders(order_number,fk_id_user,result_sum)
      VALUES (NEW.fk_order_number,NEW.fk_id_user,sum_price);
    ELSIF (EXISTS (SELECT order_number FROM orders
                   WHERE order_number = NEW.fk_order_number)) THEN
      UPDATE orders
      SET result_sum = result_sum + NEW.amount * NEW.price
      WHERE order_number = NEW.fk_order_number;
    END IF;
  ELSE
    RAISE EXCEPTION 'Количество данного товара в наличии меньше, чем
указанно в заказе';
  END IF;

    RETURN NEW;
  END;
$product_in_order_insert$ LANGUAGE plpgsql;

CREATE TRIGGER product_in_order_insert
 BEFORE INSERT
 ON product_in_order
```

```
4' FOR EACH ROW

48. EXECUTE PROCEDURE product_in_order_insert();
```

Листинг №2. Триггер на вставку в таблицу «товар в заказе».

Триггер на обновление таблицы «Товар в заказе» потребовалось создать, чтобы запретить внесение изменений в данную таблицу. Это осуществляется с помощью RAISE EXCEPTION. Триггер «update» с такими же функциями был создан для таблицы «Рейтинг».

Для удаления данных из таблицы «Товар в заказе» был написан триггер типа AFTER, необходимость его создания обусловлена тем, что товар в заказе может быть единственным, и при его удалении необходимо удалить сам заказ из таблицы «Заказ». Этот триггер представлен на листинге №3.

```
1.  CREATE FUNCTION product_in_order_delete()
2.  RETURNS TRIGGER AS $product_in_order_delete$
3.   BEGIN
4.    IF (NOT EXISTS (SELECT fk_order_number FROM product_in_order
5.                    WHERE fk_order_number = OLD.fk_order_number)) THEN
6.      DELETE FROM orders
7.      WHERE order_number = OLD.fk_order_number;
8.    ELSE
9.      UPDATE orders
10.     SET result_sum = result_sum - OLD.amount * OLD.price
11.    WHERE order_number = OLD.fk_order_number;
12.     UPDATE products
13.     SET count_product = count_product + OLD.amount
```

```
14.       WHERE id_product= OLD.fk_id_product;
15.     END IF;
16.     RETURN OLD;
17.     END;
18.   $product_in_order_delete$ LANGUAGE plpgsql;

19.   CREATE TRIGGER Product_in_order_delete
20.     AFTER DELETE ON product_in_order
21.     FOR EACH ROW
22.     EXECUTE PROCEDURE product_in_order_delete();
```

Листинг №3. Триггер удаления в таблице «Товар в заказе»

Для реализации остальных ограничений использовались CONSTRAINT, а для каскадного удаления или обновления ON DELETE CASCADE или ON UPDATE CASCADE соответственно.

2.3. Проектирование пользовательского приложения

Для того чтобы пользователь мог совершать какие-либо действия в приложении магазина, в первую очередь, он должен иметь свой аккаунт, поэтому самой первой после запуска приложения должна появляться страница «входа в аккаунт», включающая в себя поля «логин» и «пароль», в которые пользователь может ввести свои данные, чтобы авторизоваться, а также кнопки «войти» и «зарегистрироваться». Стоит предусмотреть, что человек, пользующийся данным приложением, может допустить ошибку при вводе логина или пароля и для этого случая разработать диалоговое окно, которое уведомит пользователя об ошибке.

Кроме того, должна быть создана «страница регистрации», к которой пользователь переходит после нажатия на кнопку «зарегистрироваться» и с

помощью которой он вносит информацию о себе в базу данных. Эта часть интерфейса должна включать в себя поля, соответствующие столбцам таблиц «Пользователь» и «Адрес», кнопку «зарегистрироваться», при нажатии на которою пользователь переходит на основную страницу магазина, а также поясняющую надпись вида: «Введите свои данные». Нужно учесть, что пользователь может не заполнить все поля и при вставке пустого элемента в ячейку таблицы может возникнуть exception. Поэтому в случае возникновения такой ситуации необходимо вывести подсказку: «Все поля должны быть заполнены» и не совершать вставку введенной информации в базу данных.

После успешно пройденной регистрации или входа в аккаунт пользователь оказывается на «главной странице» магазина. Для того чтобы покупатель смог совершать покупки, он должен видеть предлагаемый ассортимент, для этого необходимо отображать список товаров, причем для каждого продукта нужно указать его название и цену. Кроме того, в приложении обязательно должна быть реализована возможность добавлять товар в «корзину» с помощью нажатия на кнопку рядом с выбранным для покупки продуктом.

После того как пользователь определился со списком покупок, у него должна быть возможность перейти к «корзине», для этого следует создать отдельное окно приложения, включающее в себя:

- перечень названий продуктов, которые выбрал пользователь, нажав на кнопку «Добавить в корзину» на «главной странице» приложения
- цену и количество единиц каждого товара
- кнопки «Купить» и «Очистить»
- итоговую сумму заказа

В качестве дополнительных возможностей приложения, направленных на достижение большего удобства использования, на «главной странице» может быть добавлены поиск продуктов по заданным категориям и поисковая строка, в которой пользователь сам сможет вписать название искомого товара, вместе с кнопкой «Найти», при нажатии на которую как раз и осуществляется выборка.

3. Реализация пользовательского приложения

Для реализации пользовательского приложения был выбран язык программирования : Python.

Взаимосвязь с созданной базой данных была установлена с помощью библиотеки: psycopg2.

Создание GUI-интерфейса осуществлялось с помощью библиотеки: PyQt5.

Вначале с помощью метода connect библиотеки psycopg2, который создает соединение с экземпляром базы данных PostgreSQL и возвращает объект класса connect, было выполнено подключение к базе данных интернет-магазина. После этого был использован метод cursor() для создания объекта класса cursor, который позволяет программе Python выполнять команду PostgreSQL в сеансе базы данных. Курсоры привязаны к соединению на протяжении всего жизненного цикла, и все команды выполняются в контексте сеанса базы данных, завернутого соединением.[1]

Далее был создан объект приложения (экземпляр Qapplication), что является неотъемлемой частью написания программы на PyQt5.

3.1. Класс StartPage.

Затем был написан класс StartPage, являющийся наследником класса Qwidget и реализующий интерфейс страницы «входа в аккаунт». Для отображения на странице надписей «Логин» и «Пароль» и полей справа от них для записи данных был использован класс QformLayout, который управляет формами виджетов ввода и связанных с ними меток. В объект этого класса с помощью метода addRow() были добавлены пары, связывающие сами надписи, переданные как строка, и поля, в которых пользователь может записывать данные. Для создания таких полей был использован виджет QlineEdit, представляющий собой редактор однострочного текста. Чтобы хранить текст, введенный пользователем в эти поля, были созданы методы textLogin и

textPassword, которые с помощью функции connect() были подключены к сигналу textChanged(), испускаемому при изменении текста внутри текстового поля.

Кроме того, были созданы кнопки «Войти» и «Зарегистрироваться» с помощью виджета QpushButton. При нажатии на кнопку «Войти» испускается сигнал clicked, к которому был подключен метод clickedOnSignIn с помощью функции connect(). В этом методе проверяется существует ли логин и пароль, записанные в соответствующих текстовых полях на момент нажатия кнопки, в базе данных. Если таковые имеются, то создается объект класса ShopPage, который является наследником классов StartPage и QWidget, затем у этого объекта вызывается унаследованный метод show(), который отображает на экран виджет, описанный в классе ShopPage, (фактически происходит переключение между «окнами» приложения), затем с помощью функции self.close() выполняется закрытие основного виджета, описанного в классе StartPage, т. е. осуществляется закрытие окна, отвечающего за вход в аккаунт. Если же в базе данных в таблице «Пользователь» не найдено строки, соответствующей введенным пользователем данным, создается объект класса LoginErrorPage, который также является наследником классов StartPage и QWidget, и вызывается метод show, который отображает виджет, описанный в классе LoginErrorPage.

Проверка наличия логина и пароля, введенных пользователем, в таблице «Пользователь» базы данных интернет-магазина осуществляется с помощью метода existInDatabase. В этом методе с помощью объекта класса Cursor, созданного ранее, передает к базе данных интернет-магазина SELECT запрос, возвращающий логин и пароль, если они существуют в базе данных, иначе None. Согласно результату запроса метод возвращает значение true или false.

Все элементы виджета, описанного в классе StartPage, хранятся и отображаются в экземпляре класса QVBoxLayout, который выстраивает

виджеты в вертикальную линию. При этом кнопки «Войти» и «Зарегистрироваться» добавлены в объект QVBoxLayout в составе экземпляра класса QHBoxLayout, который располагает виджеты горизонтально.

Чтобы «окно» виджета находилось в центре экрана монитора, был написан метод center(). Его принцип работы заключается в том, что с помощью унаследованного метода frameGeometry() создается объект класса QRect, описывающий прямоугольник, размеры которого совпадают с «окном» виджета, также, используя метод QdesktopWidget().availableGeometry().center(), выясняется разрешение экрана монитора пользователя и из этого разрешения выявляется центральная точка. После этого у полученного прямоугольника вызывается метод moveCenter, который перемещает его центр в позицию центральной точки монитора, вычисленную ранее. Потом у объекта self класса StartPage вызывается унаследованный метод move, которому в качестве параметра передается объект класса QPointF, полученный вызовом метода topLeft() у экземпляра класса QRect и указывающий позицию для перемещения виджета, таким образом верхняя левая точка окна приложения перемещается в верхнюю левую точку прямоугольника. В итоге совершается центрирование «окна» виджета на экран монитора пользователя.

3.2. Класс LoginErrorPage.

В том случае, когда пользователь неправильно вводит логин и/или пароль и нажимает на кнопку «войти» появляется новое окно приложения, которое описывается в классе LoginErrorPage, являющимся наследником классов Qwidget и StartPage. Его предназначение заключается в оповещении пользователя об ошибке, поэтому интерфейс «окна» виджета включает в себя только надпись: "Логин или пароль введены неправильно, попробуйте снова", созданную с помощью виджета Qlabel. Для большей выразительности сообщения об ошибке, потребовалось изменить стиль текста и виджета с помощью вызовов метода setStyleSheet.

Для того чтобы надпись находилась в середине окна приложения, у экземпляра класса Qlabel вызывается метод setAlignment, позволяющий указать положение надписи относительно окна. Этому методу в качестве параметра передается значение Qt.AlignCenter, что означает размещение объекта в центре.

Кроме того, в классе LoginErrorPage также был определен метод center() аналогичный одноименному методу класса StartPage.

Сам виджет, описанный в классе LoginErrorPage, представляет собой QHBoxLayout, в который помещен объект класса Qlabel.

3.3. Класс RegistrationPage.

После нажатия на кнопку «Зарегистироваться» пользователь попадает в следующее окно приложения, описанное в классе RegistrationPage, который является наследником классов Qwidget и StartPage. Отображение текстовых полей, которые пользователю надо заполнить, поясняющих надписей к ним, и способ хранения введенного текста были реализованы так же как и в классе StartPage. Элементами окна приложения, описанного в классе RegistrationPage, также являются надпись: «Введите свои данные», созданная с помощью Qlabel, и кнопка «Зарегистрироваться», являющаяся объектом класса QpushButton. При нажатии на вышеуказанную кнопку испускается сигнал clicked, к которому с помощью функции connect() был подключен метод clickedOnCheckIn(). В этом методе проверяется есть ли поля, которые пользователь оставил пустыми и, если есть, к главному QVBoxLayout, в виде которого представлен виджет, описанный в классе RegistrationPage, добавляется виджет Qlabel, с который содержит в себе поясняющую надпись: «*Все поля должны быть заполнены». В случае когда при нажатии на кнопку нет полей, в которых отсутствует текст, происходит проверка наличия введенного логина в базе данных, что осуществляется с помощью метода existLogin, который работает аналогично методу existInDatabase класса StartPage, описанного выше. Если логин, введенный пользователем уже присутствует в базе данных, то в главный виджет

класса подобно предыдущей добавляется надпись: «*Пользователь с данным ником уже существует». В том случае, если введенные данные успешно прошли все проверки, перед добавлением полученной информации в базу данных для нового пользователя генерируется идентификатор с помощью встроенной функции hash, в качестве параметра которой передается логин, так как он является уникальным для каждого покупателя. После этого к базе данных посылается запрос типа INSERT и происходит добавление нового пользователя. При завершении транзакции окно виджета класса RegistrationPage закрывается, создается объект класса ShopPage, который является наследником данного классов RegistrationPage и QWidget, и у него вызывается метод show, который отображает виджет, описанный в классе ShopPage.

3.4. Класс ShopPage.

Для отображения основного окна приложения магазина был создан класс ShopPage. В нем описывается виджет, основой которого является объект класса QgridLayout(), который занимает отведенное ему место, разбивает его на строки и столбцы и помещает каждый подконтрольный виджет в соответствующую ячейку.

В самую верхнюю левую ячейку был добавлен виджет Qlabel, с помощью которого отображается надпись «Выберите категорию:», ниже был расположен виджет QscrollArea, содержащий область отображения, которую можно прокручивать, и две полосы прокрутки. При этом в виджет QscrollArea, был добавлен объект класса Qwidget, в который с помощью метода setLayout был помещен экземпляр класса QVBoxLayout, включающий в себя список категорий продуктов, каждая из которых реализована с помощью класса QpushButton. То есть пользователь видит перед собой расположенный вертикально перечень категорий продуктов, представленных в виде кнопок, который можно прокручивать вверх и вниз.

При нажатии на кнопку с соответствующей категорией, испускается сигнал clicked, к которому подключен метод, посылающий запрос SELECT к базе данных, для выборки продуктов запрашиваемой категории.

По средине сетки в нижней ячейке был добавлен самый важный, с точки зрения цели создания приложения, виджет QScrollArea, у которого был вызван разрешающий прокрутку метод setWidgetResizable со значением true в качестве параметра. Чтобы полоса прокрутки была видна всегда, также вызывается метод setVerticalScrollBarPolicy с параметром Qt.ScrollBarAlwaysOn.

Кроме того, объект класса QScrollArea содержит в себе объект класса QVBoxLayout, включающего в себя объекты класса Product, унаследованного от класса Qwidget. В конструктор класса Product передаются строки, содержащие название товара и его цену, кроме того, класс включает в себя экземпляр класса QHBoxLayout, который выстраивает виджеты горизонтально и в свою очередь включает в себя 2 кнопки, первая из которых отображает наименование и стоимость товара и не выполняет никаких функций при нажатии, переданные конструктору, а вторая содержит надпись: «Добавить в корзину». При нажатии на последнюю вызывается метод clickedOnAddBascet, в котором с помощью запроса SELECT, проверяется существует ли уже заказ данного пользователя в таблице «Заказ». Если такового не имеется, то идентификатор заказа создается с помощью функции hash, в качестве параметра которой передается идентификатор пользователя, совершающего заказ. В случае если покупатель добавляет продукт в заказ уже не в первый раз, в базу данных посылается запрос SELECT, совершающий выборку идентификатора заказа . После этого выполняется очередной запрос SELECT, чтобы по названию товара, переданному в конструктор класса, определить его идентификатор. Затем, когда все сведения об идентификаторах пользователя, заказа и товара собраны, посылается запрос SELECT в таблицу «Товар в заказе» для получения информации о количестве данного товара уже заказанном пользователем. Если

результат запроса не является пустым, то его значение записывается во вспомогательную переменную amount и увеличивается на единицу, после чего строка, соответствующая заказываемому пользователем продукту в таблице «Товар в заказе» удаляется. Последние действия необходимы, так как обновление этой таблицы запрещено бизнес-логикой базы данных.

После чего, происходит вставка данных в таблицу «Товар в заказе», где значение количества заказываемого продукта, либо равно переменной amount, либо равно единице, если товар ранее не был в заказан.

В среднюю верхнюю ячейку сетки класса ShopPage был добавлен виджет, представляющий собой объект класса QlineEdit, а в ячейку справа был помещен виджет в виде кнопки «Найти». Эти два виджета в совокупности представляют собой строку поиска для товаров в магазине. При нажатии на кнопку «Найти» вызывается метод clickedOnSearch, в котором к базе данных посылается запрос SELECT, чтобы узнать есть ли продукт с названием, введенным пользователем, в таблице «Товар». Если такой существует, то создается объект класса Product, который затем с помощью метода setWidget выбирается для отображения в виджете, расположенном в средней нижней ячейке.

В самой нижней левой ячейке сетки, располагается кнопка «Корзина», сигнал о нажатии на которую передается методу clickedOnBascet, с помощью которого осуществляется переход к окну приложения, описанному в классе BascetPage.

3.5. Класс BascetPage.

Класс BascetPage был написан для создания окна приложения, которое представляет собой «корзину» покупок пользователя. Все структурные элементы описанные в классе помещены в объект класса QVBoxLayout.

Для создания списка продуктов, добавленных пользователем в «корзину», к базе данных посылается несколько запросов SELECT , из результатов которых формируется три массива, один содержит названия продуктов, второй -

количество, а третий - цену. Затем в цикле с помощью объекта класса QgridLayout формируется сетка, в первом столбце которой находятся наименования товаров, во втором - количество, а в третьем — цена.

После этого полученный экземпляр класса QgridLayout, с помощью метода setLayout помещается в объект класса Qwidget, который в свою очередь находится в составе объекта класса QscrollArea.

Также для удобства пользователя был создан объект классса Qlabel, отражающий итоговаую сумму заказа, которая подгружается из соответствующей колонки «общая сумма» таблицы «Заказ» с помощью запросов SELECT.

Кроме того, в классе BascetPage были описаны 2 кнопки: «Купить» и «Очистить». Сигнал clicked последней из них соединен с методом deleteOrders, в котором с помощью запросов к базе данных происходит восстановление количества товара на складе магазина, то есть в таблице «Товар», после чего у объекта класса Qwidget, отвечающего за хранение сетки, в которой отображен список покупок, вызывается метод deleteLater(), который удаляет объект класса QgridLayout, а значит и перечень товаров, заказанных пользователем, также обнуляется итоговая сумма заказа.

При нажатии на кнопку «Купить» также происходит удаление экземпляр класса QgridLayout и обнуление конечной суммы заказа, но при этом какие-либо изменения в базе данных отсутствуют.

4. Тестирование

Для того чтобы удостовериться в правильности работы программы и ее успешного взаимодействия с базой данных были проведены некоторые тесты, представленные в таблице 10.

Таблица 10. Тестирование работы базы данных и приложения.

Выполненные действия	Ожидаемый результат	Соответствие полученного результата ожидаемому
В окне приложения, описанного в классе StartPage, нажатие на кнопку «Войти», когда введенные данные в поля «Логин» и/или «Пароль» существуют в соответствующих ячейках таблицы «users» в базе данных	Закрытие текущего окна приложения и открытие следующего, описанного в классе ShopPage	Полностью соответствует
В окне приложения, описанного в классе StartPage, нажатие на кнопку «Войти», когда введенные данные в поля «Логин» и/или «Пароль» отсутствуют в базе данных.	Появление окна, содержащего сообщение об ошибке вида «Логин или пароль введены неправильно»	Полностью соответствует
Нажатие на кнопку «Войти» при незаполненных полях «Логин» и/или «Пароль» в окне приложения, описанного в классе StartPage.	Появление подсказки о том, что все поля должны быть заполнены	Частично соответствует, т. к. появления подсказки заменено на

		сообщение об ошибке
В окне приложения, описанного в классе RegistrationPage нажатие на кнопку «Зарегистрироваться», когда нет пустых текстовых полей и данные, введенные пользователем в текстовое поле «Логин» уже существуют в одной из ячеек таблицы «users» базы данных	Появление подсказки о том, что пользователь с данным ником уже существует	Полностью соответствует
В окне приложения, описанного в классе RegistrationPage нажатие на кнопку «Зарегистрироваться», когда есть пустые текстовые поля и данные, введенные пользователем в текстовое поле «Логин» уже существуют в одной из ячеек таблицы «users» базы данных	1) Отправка запроса SELECT к таблице «users» 2) Появление подсказки о том, что пользователь с данным ником уже существует 3) Появление подсказки о том, что все поля должны быть заполнены	Частично соответствует, так будет выведена только подсказка о том, что все поля должны быть заполнены
В окне приложения, описанного в классе RegistrationPage нажатие на кнопку «Зарегистрироваться», когда все текстовые поля заполнены и данные, введенные	1) Вставка новых записей в таблицы «users» и «users_adress», в полях которых содержится информация, введенная пользователем	Полностью соответствует

пользователем в текстовое поле «Логин» не существуют ни в одной из ячеек таблицы «users» базы данных	2) Закрытие текущего окна приложения и открытие следующего, описанного в классе RegistrationPage	
В окне приложения, описанного в классе BascetPage, нажатие на кнопку «Очистить»	1) Восстановление количества товара на складе магазина, то есть в таблице «products», с помощью запросов к базе данных 2) Очистка списка товаров, заказанных пользователем 3) Обнуление итоговой суммы заказа	Полностью соответствует
В окне приложения, описанного в классе ShopPage, нажатие на кнопку «Добавить в корзину»	Вставка новых записи в таблицу «product_in_order», в полях которых содержится информация о выбранном пользователем продукте	Полностью соответствует
В окне приложения, описанного в классе BascetPage, нажатие на кнопку «Купить»	1) Изменение статуса заказа 2) Совершение покупки пользователем	Не соответствует, происходит очищение списка товаров в «корзине» и обнуление итоговой суммы заказа

В окне приложения, описанного в классе ShopPage, нажатие на кнопку, соответствующую одной из категорий	Отображения списка продуктов, принадлежащих выбранной категории, формируемого с помощью SELECT запроса к таблице «products» базы данных	Полностью соответствует
Вставка в таблицу «product_in_order» новой записи, где fk_order_number = 555 fk_id_user = 11 fk_id_product = 3 price = 200 amount = 10. При условии, что в записи, где значение ячейки поля id_product таблицы «products» равно идентификатору, содержащемуся в ячейке поля fk_id_product вставляемой записи, значение ячейки в поле count_product таблицы «products» меньше 10	По причине срабатывания триггера «insert» для таблицы «product_in_order», появление сообщения об ошибке вида: «Количество данного товара в наличии меньше, чем указанно в заказе»	Полностью соответствует
Вставка в таблицу «product_in_order» новой записи, где fk_order_number = 444 fk_id_user = 11	1) появление соответствующей записи в таблице «product_in_order» 2) появление в таблице «orders» новой записи,	Полностью соответствует

fk_id_product = 3 price = 200 amount = 10 При условиях: 1) в записи, где значение ячейки поля id_product таблицы «products» равно идентификатору, содержащемуся в ячейке поля fk_id_product вставляемой записи, значение ячейки в поле count_product таблицы «products» больше 10. 2) записи, где значение ячейки поля order_number таблицы «orders» равно идентификатору, содержащемуся в ячейке поля fk_ order_number вставляемой записи, не существует	где ячейки полей order_number и fk_id_user совпадают со соответствующими значениями ячеек полей fk_order_number и fk_id_user вставляемой записи, значение ячейки поля date_order, соответствует дате вставки новой записи в таблицу «product_in_order», а ячейка поля result_sum содержит произведение чисел, хранящихся в полях price и amount вставляемой записи, благодаря срабатыванию триггера «insert» для таблицы «product_in_order»	
Вставка в таблицу «product_in_order» новой записи, где fk_order_number = 555 fk_id_user = 11 fk_id_product = 3 price = 200 amount = 10. При условиях:	1) появление соответствующей записи в таблице «product_in_order» 2) в таблице «orders» в записи, в которой содержимое ячейки поля order_number совпадает с идентификатором, хранящимся в ячейке поля fk_order_number вставляемой	Полностью соответствует

1) в записи, где значение ячейки поля id_product таблицы «products» равно идентификатору, содержащемуся в ячейке поля fk_id_product вставляемой записи, значение ячейки в поле count_product таблицы «products» больше 10. 2) записи, где значение ячейки поля order_number таблицы «orders» равно идентификатору, содержащемуся в ячейке поля fk_ order_number вставляемой записи, уже существует	записи, значение находящееся в ячейке поля result_sum увеличивается на величину равную произведению чисел, хранящихся в полях price и amount вставляемой записи	
В таблице «product_in_order» в записи, fk_order_number которой равен 555, замена значения ячейки, хранящей информацию о цене товара, на «новое» равное 150 единицам.	1) из-за срабатывания триггера «update» для таблицы «product_in_order», появление сообщения об ошибке вида: «Изменение данных в таблице product_in_order запрещено»	Полностью соответствует
Удаление из таблицы «product_in_order» единственной записи, у которой значение ячейки в поле fk_order_number равно 555.	1) удаление соответствующей записи из таблицы «product_in_order» 2) удаление записи, у которой в поле order_number значение	Полностью соответствует

	ячейки равно 555, из таблицы «orders», благодаря срабатыванию триггера «delete» для таблицы «product_in_order»	
Удаление из таблицы «Товар в заказе» одной из нескольких записей, у которых значение ячейки в поле fk_order_number равно 3.	1) удаление соответствующей записи из таблицы «product_in_order» 2) в записи, где значение ячейки поля order_number таблицы «orders» равно идентификатору, содержащемуся в ячейке поля fk_order_number удаленной записи, значение ячейки в поле result_sum таблицы «orders» должно уменьшится на величину равную произведению значений ячеек в полях amount и price удаленной записи, благодаря срабатыванию триггера «delete» для таблицы «product_in_order» 3) в записи, где значение ячейки поля id_product таблицы «products» равно идентификатору, содержащемуся в ячейке поля fk_id_product удаленной	Полностью соответствует

	записи, значение ячейки в поле таблицы «orders» должно увеличиться на величину равную значению ячейки в поле count_product удаленной записи, благодаря срабатыванию триггера «delete» для таблицы «product_in_order»	
Вставка в таблицу «delivery» новой записи, где id_delivery = 125 fk_id_product = 3 fk_providers_company_name = ‘Магнит“ price,= 200 amount = 14	1) появление соответствующей записи в таблице «delivery» 2) увеличение количества добавляемого товара в таблице «products» на 14 единиц, благодаря срабатыванию триггера «insert» таблицы «delivery»	Полностью соответствует

5. Заключение

В ходе написания курсового проекта была создана база данных для интернет-магазина и написана программа пользовательского приложения для покупателей.

Был получен опыт работы с СУБД pgAdminIII и изучен синтаксис языка запросов pgSQL. Кроме того, были освоены набор библиотек для создания графического интерфейса PyQt5 и библиотека psycopg2, с помощью которой осуществляется взаимосвязьбазой данных PostgreSQL.

В дальнейшем в пользовательское приложение можно добавить интерфейс для продавцов и владельцев интернет-магазина. Также, более детально проработав, получившуюся базу данных можно подключить ее к сайту интернет-магазина.

В итоге, можно сделать вывод о том, что при дальнейшем усовершенствовании созданная база данных и пользовательское приложение могут быть полезны для создания интернет-магазина.

Список литературы.

1. Psycopg 2.7.4.dev0 documentations.

   Режим доступа: http://www.psycopg.org/psycopg/docs

2. Qt Documentation.

   Режим доступа: http://doc.qt.io/qt-5/qwidget.html

3. Справочная документация по Qt (Выпуск Open Source).

   Режим доступа: http://doc.crossplatform.ru/qt/4.5.0

4. Qt Справочная Документация (Desktop Edition).

   Режим доступа: http://qtdocs.narod.ru/4.1.0/doc/html

5. Туманов В. Основы проектирования реляционных баз данных//Лекция 1: Информационные системы с базами данных.

   Режим доступа: http://www.intuit.ru/studies/courses/1095/191/lecture/4967?page=2

6. Вишняков И. Э. Лекции по предмету «Базы данных».

   Режим доступа: https://yadi.sk/d/UOifGiyLv5K7w

7. Стасышина Т.Л. Создание триггеров в PostgreSQL.

   Режим доступа: http://postgresql.ru.net/docs/trigger.html