

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ**

**ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ  
«ДОНСКОЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»**

**(ДГТУ)**

Факультет \_\_\_\_\_\_\_\_\_\_\_\_\_Информатика и вычислительная техника\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(наименование факультета)

Кафедра Программное обеспечение вычислительной техники и автоматизированных систем

(наименование кафедры)

|  |  |  |
| --- | --- | --- |
|  | | |
| Зав. кафедрой | | «ПОВТиАС» |
| \_\_\_\_\_\_\_\_\_\_\_\_ | | В.В. Долгов |
| (подпись) | | (И.О.Ф.) |
| «\_\_\_» | \_\_\_\_\_\_\_\_\_\_\_\_\_ 2020г. | |

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

к курсовому проекту (работе) по дисциплине (модулю): \_\_\_\_\_\_\_\_\_\_\_Базы данных\_\_\_\_\_\_\_\_\_

(наименование учебной дисциплины (модуля))

на тему \_\_\_\_\_\_\_\_\_ Разработка базы данных «Магазин компьютерных комплектующих» \_

Автор проекта (работы) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Д. С. Кононов \_

подпись И.О.Ф.

Направление/специальность, профиль/специализация:

09.03.04 Программная инженерия \_

код направления наименование направления (специальности)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ПОВТиАС\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

наименование профиля (специализации)

Обозначение курсового проекта (работы) 09.03.04.420000.000 КР Группа ВПР31

Руководитель проекта \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ доцент, С.П.Новиков

подпись (должность, И.О.Ф.)

Проект (работа) защищен (а) \_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_

дата оценка подпись

Ростов-на-Дону

2020



МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ**

**ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ  
«ДОНСКОЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»**

**(ДГТУ)**

Факультет \_\_\_\_\_\_\_\_\_ Информатика и вычислительная техника \_\_\_ \_\_\_

(наименование факультета)

Кафедра Программное обеспечение вычислительной техники и автоматизированных систем (наименование кафедры)

|  |  |  |
| --- | --- | --- |
|  | | |
| Зав. кафедрой | | «ПОВТиАС» |
| \_\_\_\_\_\_\_\_\_\_\_\_ | | В.В. Долгов |
| (подпись) | | И.О.Ф. |
| «\_\_\_» | \_\_\_\_\_\_\_\_\_\_\_\_\_ 2020г. | |

**ЗАДАНИЕ**

на курсовой проект (работу)

Студент \_\_\_\_\_Кононов Д. С.\_\_\_\_\_\_\_ Код 09.03.04.420000.000 КР Группа \_ВПР31\_

Тема \_\_\_\_ Разработка базы данных «Магазин компьютерных комплектующих» \_

Срок представления проекта (работы) к защите «\_\_\_\_»\_\_\_\_\_\_\_\_\_\_ 2020 г.

Исходные данные для курсового проекта (работы)

1. Задание на выполнение курсовой работы\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

2. Лекционный материал по дисциплине «Базы данных»; Дейт К.Дж. Введение в системы баз данных, 6-е издание: Пер. с англ. – К.; М.; СПб.: Издательский дом «Вильямс», 2000. – 848 с.; Коннолли Т., Бегг К., Страчан А. Базы данных: проектирование, реализация и сопровождение. Теория и практика, 2-е изд.: Пер. с англ. – М.: Издательский дом «Вильямс», 2000. – 1120 с.; Карпова Т.С. Базы данных: модели, разработка, реализация. – СПб.: Питер, 2001. – 304 с.; М. Лутц. Изучаем Python. 4-е изд.: – СПб.: Питер, 2019. – 504 с.; Паттон Джефф. Пользовательские истории. Искусство гибкой разработки ПО. – СПб.: Питер, 2019. – 288 с.

|  |  |  |  |
| --- | --- | --- | --- |
| Содержание пояснительной записки | | | |
| ВВЕДЕНИЕ: | | | |
| В разделе «Введение» описаны основные понятия, используемые в базах данных, а также их характеристика. Была поставлена задача разработки базы данных магазина компьютерных комплектующих. Приведены краткие описания каждой главы. | | | |
| Разделы основной части:  1. Анализ предметной области и проектирование структуры базы данных. В данном разделе описывается выбранная предметная область: приводятся основные особенности работы магазина компьютерных комплектующих, выделяются конечные пользователи и описываются их основные функции. Оформление модели уровня представлений.  2. Физическая реализация разработанной модели. В данном разделе описывается проектирование таблиц в среде целевой СУБД. Описание основных функций для работы с базой данных.  3. Разработка клиентского интерфейса для работы с базой данных. Приводится описание основных классов приложения. Обзор методов клиентского интерфейса. Тестирование приложения. | | | |
| ЗАКЛЮЧЕНИЕ: | | | |
| В «Заключении» подведены итоги курсовой работы с описанием решения поставленной задачи, описание разработанной программы интерфейса для различных пользователей магазина компьютерных комплектующих.  Перечень графического материала   1. Рисунок1 – Создание БД «PC\_shop» 2. Рисунок 2 – Создание таблицы «components» 3. Рисунок 3 – Триггер и триггерная функция hcomponents 4. Рисунок 4 – Авторизация пользователя 5. Рисунок 5 – Окно для работы с БД для менеджера 6. Рисунок 6 – Окно для работы с БД для клиента 7. Рисунок 7 – Окно для работы с БД для кладовщика 8. Рисунок 8 – Поиск по значению 9. Рисунок 9 – Резервное копирование и восстановление 10. Приложение А – ER-диаграмма 11. Приложение Б – Даталогическая таблица 12. Приложение В – UML-диаграмма | | | |
|  | | | |
|  | | | |
| Руководитель проекта (работы) | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  подпись, дата | С. П.Новиков  \_  И.О.Ф. |
|  |  |  |
| Задание принял к исполнению | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  подпись, дата | \_\_\_\_\_Д. С. Кононов \_  И.О.Ф. |

СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 5](#_Toc34771350)

[1 Анализ предметной области и проектирование структуры базы данных 7](#_Toc34771351)

[1.1 Формирование уровня представлений базы данных магазина компьютерных комплектующих 7](#_Toc34771352)

[1.1.1 Описание уровня представлений пользователей и выделение основных сущностей 7](#_Toc34771353)

[1.1.2 Описание сущностей, их атрибутов и доменов 8](#_Toc34771354)

[1.1.3 Создание схемы в виде ER – диаграммы 14](#_Toc34771355)

[1.1.4 Описания связей между объектами 16](#_Toc34771356)

[1.2 Построение концептуального уровня базы данных магазина компьютерных комплектующих 17](#_Toc34771357)

[1.2.1 Описание функциональных зависимостей, имеющих место в предметной области 17](#_Toc34771358)

[1.2.2 Нормализация отношений 19](#_Toc34771359)

[1.2.2.1 Нормализация до первой нормальной формы 19](#_Toc34771360)

[1.2.2.2 Нормализация до второй нормальной формы 20](#_Toc34771361)

[1.2.2.3 Приведение к третьей нормальной форме. Выводы 21](#_Toc34771362)

[1.2.3 Логическое (даталогическое) проектирование базы данных 22](#_Toc34771363)

[2 Физическая реализация модели базы данных магазина компьютерных комплектующих 24](#_Toc34771364)

[2.1 Обоснования выбора целевой СУБД 24](#_Toc34771365)

[2.2 Создание таблиц данных в среде целевой СУБД 28](#_Toc34771366)

[3 Разработка клиентского интерфейса для работы с базой данных 31](#_Toc34771367)

[3.1 Выбор среды для разработки приложения 31](#_Toc34771368)

[3.2 Описание основных классов и методов интерфейса 34](#_Toc34771369)

[3.3 Реализация создания резервной копии и восстановления базы данных 36](#_Toc34771370)

[3.4 Пример работы клиентского интерфейса 36](#_Toc34771371)

[ЗАКЛЮЧЕНИЕ 39](#_Toc34771372)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 40](#_Toc34771373)

[ПРИЛОЖЕНИЕ А – ER-диаграмма 41](#_Toc34771374)

[ПРИЛОЖЕНИЕ Б – Даталогическая таблица 42](#_Toc34771375)

[ПРИЛОЖЕНИЕ В – UML-диаграмма 43](#_Toc34771376)

[ПРИЛОЖЕНИЕ Г – Скрипт базы данных 44](#_Toc34771377)

[ПРИЛОЖЕНИЕ Д – Код клиентского интерфейса 50](#_Toc34771378)

[ПРИЛОЖЕНИЕ Е – Отзыв руководителя 54](#_Toc34771379)

ВВЕДЕНИЕ

На сегодняшний день большинство приложений так или иначе используют структурированную информацию или, другими словами, упорядоченные данные. Понятие информации - любых сведений о каком-либо событии, процессе или объекте - является одним из важнейших понятий в теории баз данных.

База данных (БД) — это информационная модель, позволяющая в упорядоченном виде хранить данные о группе объектов, обладающих одинаковым набором свойств [1]. БД состоит из множества связанных файлов и используется в целях адекватного отображения изменяющегося реального мира и удовлетворения информационных потребностей пользователей. Данные запоминаются так, чтобы они были независимы от программ, использующих эти данные. Для добавления новых или модификации существующих данных, а также для поиска данных в БД применяется система управления базами данных (СУБД).

БД всегда были важнейшей темой при изучении информационных систем. Однако в последние годы всплеск популярности Интернета и бурное развитие новых технологий для Интернета сделали знание технологии баз данных для многих одним из актуальнейших путей карьеры. Технологии баз данных увели интернет-приложения далеко от простых брошюрных публикаций, которые характеризовали ранние приложения. В то же время интернет-технология обеспечивает пользователям стандартизированные и доступные средства публикации содержимого баз данных. Правда, ни одна из этих новых разработок не отменяет необходимости в классических приложениях баз данных, которые появились еще до развития интернета для нужд бизнеса. Это только расширяет важность знания баз данных.

СУБД – это программное обеспечение, предназначенное для создания, ведения и совместного использования БД многими пользователями. Простым языком, система управления базами данных выступает в качестве посредника между базой данных и ее пользователями [1].

Предметная область – некоторая часть реального мира, подлежащая изучению с целью организации управления и последующей автоматизации. Предметная область представляется множеством фрагментов, которые, в свою очередь, характеризуются множеством объектов и процессов, использующих эти объекты, а также конечным множеством пользователей, характеризуемых различными взглядами на предметную область.

Цель базы данных дать возможность организациям вести учет склада и покупок магазина компьютерных комплектующих. В мире существует множество систем управления базами данных. Несмотря на то, что они могут по-разному работать с разными объектами и предоставляют пользователю различные функции и средства, большинство СУБД опираются на единый устоявшийся комплекс основных понятий. Это дает нам возможность рассмотреть одну систему и обобщить ее понятия, приемы и методы на весь класс СУБД.

В первой главе выполнен анализ предметной области, выделены основные сущности, а также описана база данных на внешнем и концептуальном уровнях.

Во второй главе описана реализация базы данных магазина компьютерных комплектующих на физическом уровне.

В третьей главе описана реализация клиентского интерфейса и рассмотрена работа приложения на контрольном примере.

**1 Анализ предметной области и проектирование структуры базы данных**

При проектировании базы данных процесс перехода от реальности к информационной модели происходит в несколько этапов.

1.1 Формирование уровня представлений базы данных магазина компьютерных комплектующих

1.1.1 Описание уровня представлений пользователей и выделение основных сущностей

В качестве предметной области выбирается магазин компьютерных комплектующих, для которого необходима база данных списка заказов и самих компьютерных комплектующих. С точки зрения проектирования базы данных в данной предметной области имеются следующие конечные пользователи: клиент, менеджер и кладовщик. Опишем предметную область с точки зрения конечных пользователей.

**Кладовщик**:в своей работе ему потребуется постоянно иметь возможность просмотра и изменения количества товаров на складе, важным аспектом работы кладовщика является отгрузка товаров, которые купил клиент, и так же оперативно вносить информацию в базу данных, и так же эффективно использовать её во время работы. Кладовщик имеет доступ к следующим сущностям: склад, заказ.

**Менеджер**: ему необходима возможность в любое время просматривать информацию о комплектующих, клиентах и заказах, также возможность закрывать заказы и передавать списки товаров для отгрузки со склада кладовщику необходимого склада. Изменять и добавлять комплектующие. Менеджер имеет доступ к следующим сущностям: клиент, заказ, корзина, комплектующая.

**Клиент**: ему необходимо иметь доступ к списку комплектующих и их характеристикам, остатку на складе, и так же возможность добавлять эти товары в корзину и отправлять заказ на обработку менеджеру. Клиент имеет доступ к следующим сущностям: комплектующая, заказ, корзина.

Из данных описаний предметной области выделим базовые объекты: Клиент, Корзина, Комплектующая, Заказ, Должность, Склад, Брэнд, Работник, Склады.

Построенная ER-модель находиться в приложении А, а построенная даталогическая схема в приложении Б.

1.1.2 Описание сущностей, их атрибутов и доменов

Объект «Клиент»:

clients = {

id: Integer;

name: Varchar (30);

phone\_number: Integer;

discount\_card: Integer;

login: Varchar (32);

password: Varchar (32);

}

Объект «Корзина»:

basket = {

id: Integer;

id\_component: значение «id» отношения «components»;

quantity: Integer;

id\_client: значение «id» отношения «clients»;

id\_order: значение «id» отношения «orders»;

}

Объект «Комплектующая»:

components = {

id: Integer;

name: Varchar (30);

puтрока (3s0rd: XBrchase price: Integer;

selling price: Integer;

clock: Integer;

cores: Integer;

threads: Integer;

clock memory: Integer;

bus width: Integer;

memory: Integer;

timing: Integer;

id\_brand: значение «id» отношения «brands»;

}

Объект «Заказ»:

orders = {

id: Integer;

data\_order: Date;

id\_manager: значение «id» отношения «workers»;

discount: Integer;

}

Объект «Должность»:

positions = {

id: Integer;

name: Varchar (30);

}

Объект «Склад»:

stock = {

id: Integer;

id\_component: значение «id» отношения «components»;

id\_stock: значение «id» отношения «stocks»;

balance: Integer;

}

Объект «Брэнд»:

brands = {

id: Integer;

name: Varchar (30);

}

Объект «Работник»:

workers = {

id: Integer;

name: Varchar (30);

address: Varchar (50);

phone\_number: Integer;

id\_position: значение «id» отношения «positions»;

e-mail: Varchar (30);

data\_employment: Date;

}

Объект «Склады»:

stocks = {

id: Integer;

name: Varchar (30);

inn: Integer;

ogrn: Integer;

address: Varchar (50);

data create: Date;

id\_storekeeper: значение «id» отношения «workers»;

}

Описание доменов магазина компьютерных комплектующих:

|  |  |
| --- | --- |
| Имя атрибута | Домен атрибута |
| id | Int |
| name | String(30) |
| phone\_number | Int |
| discount\_card | Int |
| login | String(32) |
| password | String(32) |

Таблица 1.1 – Отношение «clients»

Ограничения:

id – первичный ключ (уникальное значение)

|  |  |
| --- | --- |
| Имя атрибута | Домен атрибута |
| id | Int |
| id\_component | Int |
| quantity | Int |
| id\_client | Int |
| id\_order | Int |

Таблица 1.2 – Отношение «basket»

Ограничения:

id – первичный ключ (уникальное значение)

id\_component – внешний ключ к отношению «components» поле «id», обновление каскадное, удаление каскадное.

id\_client – внешний ключ к отношению «clients» поле «id», обновление каскадное, удаление каскадное.

id\_order – внешний ключ к отношению «orders» поле «id», обновление каскадное, удаление каскадное.

|  |  |
| --- | --- |
| Имя атрибута | Домен атрибута |
| id | Int |
| name | String(30) |
| purchase\_price | Int |
| selling\_price | Int |
| Clock | Int |
| Cores | Int |
| Threads | Int |
| clock\_memory | Int |
| bus\_width | Int |
| memory | Int |
| timing | Int |
| id\_brand | Int |

Таблица 1.3 – Отношение «components»

Ограничения:

id – первичный ключ (уникальное значение)

id\_brand – внешний ключ к отношению «brands» поле «id», обновление каскадное, удаление каскадное.

|  |  |
| --- | --- |
| Имя атрибута | Домен атрибута |
| id | Int |
| data\_order | Date |
| id\_manager | Int |
| discount | Int |

Таблица 1.4 – Отношение «orders»

Ограничения:

id – первичный ключ (уникальное значение)

id\_manager – внешний ключ к отношению «workers» поле «id», обновление каскадное, удаление каскадное.

|  |  |
| --- | --- |
| Имя атрибута | Домен атрибута |
| id | Int |
| name | String(30) |

Таблица 1.5 – Отношение «positions»

Ограничения:

id – первичный ключ (уникальное значение)

|  |  |
| --- | --- |
| Имя атрибута | Домен атрибута |
| id | Int |
| id\_component | Int |
| id\_stock | Int |
| balance | Int |

Таблица 1.6 – Отношение «stock»

Ограничения:

id – первичный ключ (уникальное значение)

id\_component – внешний ключ к отношению «components» поле «id», обновление каскадное, удаление каскадное.

id\_stock – внешний ключ к отношению «stocks» поле «id», обновление каскадное, удаление каскадное.

|  |  |
| --- | --- |
| Имя атрибута | Домен атрибута |
| id | Int |
| name | String(50) |
| inn | Int |
| ogrn | Int |
| address | String(50) |
| data\_create | Int |
| id\_storekeeper | Int |

Таблица 1.7 – Отношение «stocks»

Ограничения:

id – первичный ключ (уникальное значение)

id\_storekeeper – внешний ключ к отношению «workers» поле «id», обновление каскадное, удаление каскадное.

|  |  |
| --- | --- |
| Имя атрибута | Домен атрибута |
| id | Int |
| name | String(30) |

Таблица 1.8 – Отношение «brands»

Ограничения:

id – первичный ключ (уникальное значение)

|  |  |
| --- | --- |
| Имя атрибута | Домен атрибута |
| id | Int |
| name | String(20) |
| address | String(50) |
| phone\_number | Int |
| id\_position | Int |
| e-mail | String(30) |
| data\_employment | Date |

Таблица 1.9 – Отношение «workers»

Ограничения:

id – первичный ключ (уникальное значение).

id\_position – внешний ключ к отношению «positions» поле «id», обновление каскадное, удаление каскадное.

1.1.3 Создание схемы в виде ER – диаграммы

Важным этапом проектирования базы данных является построение логической схемы. Логическое проектирование прежде всего связано с попыткой представления предметной области в модели базы данных. В данный момент самой удобной моделью представления является модель «сущность – связь» или, как ее еще называют, ER-диаграмма.

Ниже приведены основные понятия ER-диаграмм на примере базы данных магазин компьютерных комплектующих.

Сущность – класс однотипных объектов, имеющих одинаковые свойства, информацию о которых необходимо хранить в базе данных [3].

Сущности должны быть различимы. Например, сущностью БД магазин компьютерных комплектующих является склад товаров; не может быть двух абсолютно идентичных складов.

Экземпляр сущности – конкретный представитель данной сущности, то есть сотрудник с определенной датой трудоустройства, именем и адресом проживания и так далее.

Атрибут сущности – некая поименованная характеристика сущности.

Связь – ассоциирование нескольких сущностей. Сущность также может ассоциироваться сама с собой. Наличие связей между сущностями позволяет пользователям отыскать одну запись через другую.

Связи могут быть следующих типов:

- один-к-одному, т.е., один экземпляр первой сущности связан с одним экземпляром второй сущности;

- один-ко-многим означает, что один экземпляр первой сущности связан с несколькими экземплярами второй сущности. Например, на одном складе находится много видов компьютерных комплектующих;

- многие-ко-многим означает, что каждый экземпляр первой сущности может быть связан с несколькими экземплярами второй сущности, и каждый экземпляр второй сущности может быть связан с несколькими экземплярами первой сущности [1].

Итак, на основе представлений конечных пользователей БД предприятия можно составить обобщенное представление предметной области, выделив основные объекты будущей базы данных и отбросив второстепенные. В качестве концептуальной модели была выбрана ER-диаграмма. ER-диаграммы удобны тем, что процесс выделения сущностей, атрибутов и связей является цикличным. Разработав первый приближенный вариант диаграмм, разработчик базы данных уточняет их, опрашивая экспертов предметной области – будущих конечных пользователей. ER-модель использует графическое изображение сущностей предметной области, их свойств (атрибутов), и взаимосвязей между сущностями.

Главными сущностями БД магазин компьютерных комплектующих будут следующие: «Комплектующая», «Брэнд», «Склад», «Склады», «Работник», «Должность», «Заказ», «Корзина», «Клиент».

Рассмотрим атрибуты каждой сущности, необходимые конечным пользователям для работы:

«Комплектующая»: название, бренд, закупочная цена, конечная цена, частота ядра, количество ядер, количество потоков, частота памяти, объём памяти, разрядность шины памяти, тайминг;

«Брэнд»: название;

«Склад»: товар, остаток, месторасположение склада;

«Склады»: название, ИНН, ОГРН, адрес, дата образования;

«Работник»: имя, адрес проживания, дата трудоустройства, e-mail, номер телефона, занимаемая должность;

«Заказ»: дата заказа, скидка, СНИЛС, ИНН;

«Должность»: название;

«Корзина»: товар, количество, номер заказа, клиент;

«Клиент»: имя, номер телефона, скидка, логин, пароль.

Основные связи в проектируемой ER-диаграмме: «Комплектующая» находится на «Складе»; «Склад» находится в «Складах»; «Склады» имеют «Работников»; «Работник» имеет «Должность»; «Заказ» оформляет «Работник»; «Корзина» имеет «Заказ»; «Корзина» содержит «Клиента»; «Корзина» содержит «Комплектующую».

Таким образом, на основе приведенных представлений пользователей, выделенных основных объектов, описанных атрибутов и связей была спроектирована семантическая модель предметной области «магазин компьютерных комплектующих» (Приложение А: ER-диаграмма).

1.1.4 Описания связей между объектами

1. Комплектующие находятся на складах. Отсюда возникает связь: components – stock (1: n);
2. По одному адресу может находиться несколько складов. Отсюда возникает связь: stocks – stock (1: n);
3. На складе работает кладовщик, и он может работать на нескольких складах. Отсюда возникает связь: workers - stocks (1: n);
4. У комплектующей есть бренд, и от одного бренда может быть несколько комплектующих. Отсюда возникает связь: brands – components (1: n);
5. Сотрудник имеет должность, и на одной должности может быть несколько сотрудников. Отсюда возникает связь: positions – workers (1:n);
6. Сотрудник оформляет заказ, и он может оформлять множество заказов. Возникает связь: workers – orders (1:n);
7. В корзине находятся комплектующие. Возникает связь: components - basket (1:n);
8. Запись в корзине имеет номер заказа, и записей может быть несколько. Возникает связь: orders – basket (1:n);

1.2 Построение концептуального уровня базы данных магазина компьютерных комплектующих

1.2.1 Описание функциональных зависимостей, имеющих место в предметной области

Пусть R — отношение. X и Y некоторые подмножество множества его атрибутов. Тогда Y функционально зависимо от Х. Тогда и только тогда, когда каждому элементу множества Х соответствует один и только один элемент множества Y. Функциональную зависимость записывают так: X →Y. Иначе: Y функционально зависит от X, если два кортежа отношения R совпадают по множеству атрибутов Х, то они совпадают по множеству атрибутов Y [1].

Отношение clients = {id, name, phone\_number, discount\_card, login, password}

{id}  {name, phone\_number}

{id}  {discount\_card}

{id}  {login, password}

Отношение basket = {id, id\_ component, quantity, id\_client, id\_order}

{id}  {id\_ component, quantity }

{id}  {id\_client, id\_order}

Отношение components = {id, name, puтрока (3s0rd: XBrchase\_price, selling\_price, clock, cores, threads, clock\_memory, bus\_width, memory, timing, id\_brand}

{id}  {name, puтрока (3s0rd: XBrchase\_price, selling\_price}

{id}  {clock, cores, threads, clock\_memory, bus\_width, memory, timing}

{id}  {id\_brand}

Отношение orders = {id, data\_order, id\_manager, discount}

{id}  {data\_order, id\_manager}

{id}  {discount}

Отношение positions = {id, name}

{id}  {name}

Отношение stock = {id, id\_component, id\_ stock, balance}

{id}  {id\_component, id\_stock}

{id}  {balance}

Отношение brands = {id, name}

{id}  {name}

Отношение workers = {id, name, address, phone\_number, id\_position, e-mail, data\_employment}

{id}  {name, address, phone\_number, e-mail}

{id}  {id\_position, data\_employment}

Отношение stocks = {id, name, inn, ogrn, address, data \_create, id\_storekeeper}

{id}  {name, inn, ogrn, address, data \_create}

{id}  {id\_storekeeper}

1.2.2 Нормализация отношений

1.2.2.1 Нормализация до первой нормальной формы

Отношение находится в первой нормальной форме(1НФ), если каждый его атрибут атомарен, т.е. содержит только одно значение, а не список значений [1].

Рассмотрим базовые отношения базы данных магазина компьютерных комплектующих:

Отношение **clients**(1) = {id, name, phone\_number, discount\_card, login, password}

Отношение **basket**(1) = {id, id\_ component, quantity, id\_client, id\_order}

Отношение **components**(1) = {id, name, puтрока (3s0rd: XBrchase\_price, selling\_price, clock, cores, threads, clock\_memory, bus\_width, memory, timing, id\_brand}

Отношение **orders**(1) = {id, data\_order, id\_manager, discount}

Отношение **positions**(1) = {id, name}

Отношение **stock**(1) = {id, id\_component, id\_ stock, balance}

Отношение **brands**(1) = {id, name}

Отношение **workers**(1) = {id, name, address, phone\_number, id\_position, e-mail, data\_employment}

Отношение **stocks**(1) = {id, name, inn, ogrn, address, data \_create, id\_storekeeper}

Все базовые отношения находятся в 1НФ

1.2.2.2 Нормализация до второй нормальной формы

Отношение находится во второй нормальной форме(2НФ), если все атрибуты, за исключением ключа, зависят от ключа полностью, а не частично. Иными словами, если отношение находится в первой нормальной форме, и при этом любой её атрибут, не входящий в состав первичного ключа, функционально полно зависит от первичного ключа. Функционально полная зависимость означает, что атрибут функционально зависит от всего первичного составного ключа, но при этом не находится в функциональной зависимости от какой-либо из входящих в него атрибутов (частей) [1]. Или другими словами: в 2НФ нет не ключевых атрибутов, зависящих от части составного ключа (+ выполняются условия 1НФ).

Рассмотрим выбранные первичные ключи отношений базы данных магазина компьютерных комплектующих.

Отношение «clients» имеет потенциальные ключи: {id}, {login}; целесообразнее выбрать в качестве первичного ключа атрибут {id}.

Отношение «basket» содержит один потенциальный ключ: {id} соответственно он же и является первичным.

Отношение «components» содержит один потенциальный ключ: {id} соответственно он же и является первичным.

Отношение «orders» содержит один потенциальный ключ: {id} соответственно он же и является первичным.

Отношение «positions» содержит один потенциальный ключ: {id} соответственно он же и является первичным.

Отношение «stock» содержит один потенциальный ключ: {id} соответственно он же и является первичным.

Отношение «stocks» имеет потенциальные ключи: {id}, {inn}, {ogrn}; целесообразнее выбрать в качестве первичного ключа атрибут {id}.

Отношение «brands» содержит один потенциальный ключ: {id} соответственно он же и является первичным.

Отношение «workers» содержит один потенциальный ключ: {id} соответственно он же и является первичным.

На основе рассмотренных ранее функциональных зависимостей и первичных ключей, можно утверждать, что все базовые отношения базы данных магазина компьютерных комплектующих находятся в 2НФ, т.к. первичные ключи не являются составными.

Следовательно,

Отношение **clients**(2) = {id, name, phone\_number, discount\_card, login, password}

Отношение **basket**(2) = {id, id\_ component, quantity, id\_client, id\_order}

Отношение **components**(2) = {id, name, puтрока (3s0rd: XBrchase\_price, selling\_price, clock, cores, threads, clock\_memory, bus\_width, memory, timing, id\_brand}

Отношение **orders**(2) = {id, data\_order, id\_manager, discount}

Отношение **positions**(2) = {id, name}

Отношение **stock**(2) = {id, id\_component, id\_ stock, balance}

Отношение **brands**(2) = {id, name}

Отношение **workers**(2) = {id, name, address, phone\_number, id\_position, e-mail, data\_employment}

Отношение **stocks**(2) = {id, name, inn, ogrn, address, data \_create, id\_storekeeper}

1.2.2.3 Приведение к третьей нормальной форме. Выводы

Отношение находится в третьей нормальной форме(3НФ), если оно находится во второй нормальной форме, и не имеет транзитивной зависимости атрибута от ключа. Иными словами, так как все атрибуты зависят от ключа, то отношение, чтобы считаться приведенным к третьей нормальной форме, не должно иметь зависимостей между атрибутами. [1]

Все базовые отношения базы данных магазина компьютерных комплектующих находятся в 3НФ, т.к. все не ключевые атрибуты этих отношений не транзитивно зависят от ключей.

Таким образом:

Отношение **clients**(3) = {id, name, phone\_number, discount\_card, login, password}

Отношение **basket**(3) = {id, id\_component, quantity, id\_client, id\_order}

Отношение **components**(3) = {id, name, puтрока (3s0rd: XBrchase\_price, selling\_price, clock, cores, threads, clock\_memory, bus\_width, memory, timing, id\_brand}

Отношение **orders**(3) = {id, data\_order, id\_manager, discount}

Отношение **positions**(3) = {id, name}

Отношение **stock**(3) = {id, id\_component, id\_stock, balance}

Отношение **brands**(3) = {id, name}

Отношение **workers**(3) = {id, name, address, phone\_number, id\_position, e-mail, data\_employment}

Отношение **stocks**(3) = {id, name, inn, ogrn, address, data\_create, id\_storekeeper}

Итак, была произведена нормализация отношений: структура базы данных магазина компьютерных комплектующих была приведена к виду, обеспечивающему минимальную логическую избыточность. Была сокращена потенциальная противоречивость хранимой в базе данных информации.

1.2.3 Логическое (даталогическое) проектирование базы данных

На этапе логического проектирования разрабатывается логическая структура будущей базы данных магазина компьютерных комплектующих.

База данных создаётся на основании схемы базы данных. Инфологи­ческую модель данных, построенную в виде ER-диаграммы, необходимо пре­образовать в схему БД. Преобразование ER-диаграммы в схему БД выпол­няется путем сопоставления каждой сущности и каждой связи, имеющей атрибуты, в таблицы-отношения [3].

Для реляционной модели данных даталогическая модель — это набор схем отношений, обычно с указанием первичных ключей и связей между отношениями, представляющих собой внешние ключи.

На этапе логического проектирования учитывается специфика конкретной модели данных, но может не учитываться специфика СУБД.

Ранее были описаны связи между отношениями, обоснован выбор первичных ключей. Результат логического проектирования отображен на даталогической схеме БД магазина компьютерных комплектующих (Приложение Б: Даталогическая схема).

2 Физическая реализация модели базы данных магазина компьютерных комплектующих

2.1 Обоснования выбора целевой СУБД

В мире существует множество систем управления базами данных.

Но рассмотрим основные требования к СУБД, которые необходимы для реализации базы данных магазина компьютерных комплектующих:

1. Непротиворечивость данных. Такое требование называется требованием целостности. Целостность базы данных подразумевает поддержание полной, непротиворечивой и адекватно отражающей предметную область информации.

2. Актуальность хранимых данных. В любой момент времени информация, содержащаяся в БД, должна быть современной.

3. Многоаспектное использование данных - поступление информации из различных источников в единую БД и возможность ее использования любым отделом предприятия в соответствии с правами доступа и функциями.

4. Возможность модификации системы - возможность ее расширения и модификации данных, а также дополнение новыми функциями без ущерба для системы в целом.

5. Надежность - целостность БД не должна нарушаться при технических сбоях.

6. Скорость доступа - обеспечение быстрого доступа к требуемой информации.

СУБД осуществляют взаимодействие между БД и пользователями системы, а также между БД и прикладными программами, реализующими определенные функции обработки данных.

СУБД обеспечивают надежное хранение больших объемов данных сложной структуры во внешней памяти компьютера и эффективный доступ к ним.

К основным функциям СУБД относятся:

* непосредственное управление данными во внешней и оперативной памяти и обеспечение эффективного доступа к ним в процессе решения задач;
* поддержание целостности данных и управление транзакциями;
* ведение системного журнала изменений в БД для обеспечения восстановления БД после технического или программного сбоя;
* реализация поддержки языка описания данных и языка запросов;
* обеспечение безопасности данных;
* обеспечение параллельного доступа к данным нескольких пользователей.

Существует множество СУБД для реализации физической модели базы данных, например, ﻿SQLite, MySQL и PostgreSQL.

Рассмотрим преимущества и недостатки каждой из СУБД.

SQLite - легко встраиваемая в приложения база данных. Так как это система базируется на файлах, то она предоставляет довольно широкий набор инструментов для работы с ней, по сравнению с сетевыми СУБД. При работе с этой СУБД обращения происходят напрямую к файлам, вместо портов и сокетов в сетевых СУБД. Именно поэтому SQLite очень быстая, а также мощная благодаря технологиям обслуживающих библиотек.

Преимущества SQLite:

* файловая структура - вся база данных состоит из одного файла, поэтому её очень легко переносить на разные машины;
* используемые стандарты - хотя может показаться, что эта СУБД примитивная, но она использует SQL. Некоторые особенности опущены (RIGHT OUTER JOIN или FOR EACH STATEMENT), но основные все-таки поддерживаются;
* отличная при разработке и тестировании - в процессе разработки приложений часто появляется необходимость масштабирования. SQLite предлагает всё что необходимо для этих целей, так как состоит всего из одного файла и библиотеки, написанной на языке C.

Недостатки SQLite:

* отсутствие системы пользователей - более крупные СУБД включают в свой состав системы управления правами доступа пользователей. Обычно применения этой функции не так критично, так как эта СУБД используется в небольших приложениях;
* отсутствие возможности увеличения производительности - опять, исходя из проектирования, довольно сложно выжать что-то более производительное из этой СУБД.

MySQL — это самая распространенная полноценная серверная СУБД. MySQL очень функциональная, свободно распространяемая СУБД, которая успешно работает с различными сайтами и веб приложениями. Несмотря на то, что в ней не реализован весь SQL функционал, MySQL предлагает довольно много инструментов для разработки приложений. Так как это серверная СУБД, приложения для доступа к данным, в отличии от SQLite работают со службами MySQL.

Преимущества MySQL:

* простота в работе - установить MySQL довольно просто. Дополнительные приложения, например GUI, позволяет довольно легко работать с БД;
* богатый функционал - MySQL поддерживает большинство функционала SQL;
* безопасность - большое количество функций, обеспечивающих безопасность, которые поддерживается по умолчанию;
* масштабируемость - MySQL легко работает с большими объемами данных и легко масштабируется;
* скорость - упрощение некоторых стандартов позволяет MySQL значительно увеличить производительность.

Недостатки MySQL:

* известные ограничения - по задумке в MySQL заложены некоторые ограничения функционала, которые иногда необходимы в особо требовательных приложениях;
* проблемы с надежностью - из-за некоторых способов обработки данных MySQL (связи, транзакции, аудиты) иногда уступает другим СУБД по надежности;
* медленная разработка - хотя MySQL технически открытое ПО, существуют жалобы на процесс разработки.

PostgreSQL – имеет наиболее широкий функционал из всех трех рассмотренных нами СУБД. Она свободно распространяемая и максимально соответствует стандартам SQL. PostgreSQL или Postgres стараются полностью применять ANSI/ISO SQL стандарты своевременно с выходом новых версий. От других СУБД PostgreSQL отличается поддержкой востребованного объектно-ориентированного и/или реляционного подхода к базам данных. Например, полная поддержка надежных транзакций, т.е. атомарность, последовательность, изоляционность, прочность. Благодаря мощным технологиям PostgreSQL очень производительна. Параллельность достигнута не за счет блокировки операций чтения, а благодаря реализации управления многовариантным параллелизмом, что также обеспечивает соответствие ACID. PostgreSQL очень легко расширять своими процедурами, которые называются хранимые процедуры. Эти функции упрощают использование постоянно повторяемых операций.

Достоинства PostgreSQL:

* открытое ПО, соответствующее стандарту SQL - PostgreSQL - бесплатное ПО с открытым исходным кодом. Эта СУБД является очень мощной системой;
* большое количество дополнений - несмотря на огромное количество встроенных функций, существует очень много дополнений, позволяющих разрабатывать данные для этой СУБД и управлять ими;
* расширения - существует возможность расширения функционала за счет сохранения своих процедур;
* объектность - PostrgreSQL это не только реляционная СУБД, но также и объектно-ориентированная с поддержкой наследования и много другого.

Недостатки PostgreSQL:

* производительность - при простых операциях чтения PostgreSQL может значительно замедлить сервер и быть медленнее своих конкурентов, таких как MySQL;
* популярность - по своей природе, популярностью эта СУБД похвастаться не может, хотя и присутствует довольно большое сообщество;
* хостинг - в силу вышеперечисленных факторов иногда довольно сложно найти хостинг с поддержкой этой СУБД.

Для реализации физической модели была выбрана СУБД PostgreSQL, так как она соответствует всем требования и имеет ряд важных преимуществ по сравнению с остальными СУБД.

2.2 Создание таблиц данных в среде целевой СУБД

Код базы данных в PostgreSQL представлен в Приложении Г.

В среде администрирования и разработки DataGrip с помощью SQL-запросов была создана база данных «PC\_shop»:

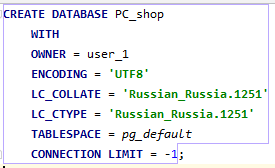


Рисунок 1 – Создание БД «PC\_shop»

А также необходимые таблицы: «Brands», «Components», «Clients», «Positions», «Workers», «Stocks», «Stock», «Orders», «Basket».

Создание таблиц представлено на примере таблицы «components»:

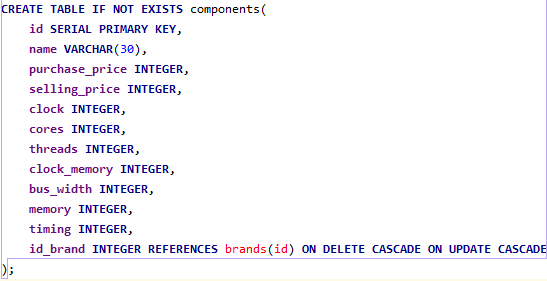


Рисунок 2 – Создание таблицы «components»

При работе с клиентским приложением необходимо вести историю изменения данных. Для этого необходимо прослеживать изменения данных для каждой таблицы. Поэтому были реализованы триггеры.

Триггер — хранимая процедура особого типа, которую пользователь не вызывает непосредственно, а исполнение которой обусловлено действием по модификации данных: добавлением INSERT, удалением DELETE строки в заданной таблице, или изменением UPDATE данных в определённом столбце заданной таблицы реляционной базы данных. Триггеры применяются для обеспечения целостности данных и реализации сложной бизнес-логики. Триггер запускается сервером автоматически при попытке изменения данных в таблице, с которой он связан. Все производимые им модификации данных рассматриваются как выполняемые в транзакции, в которой выполнено действие, вызвавшее срабатывание триггера. Соответственно, в случае обнаружения ошибки или нарушения целостности данных может произойти откат этой транзакции.

Триггер срабатывает при изменении данных в таблице «components», после записывает тип и название таблицы в таблицу «history». А также старые данные сохраняет в таблицу «hcomponents». Триггеры такого типа реализованы для всех основных таблиц базы данных магазина компьютерных комплектующих, чтобы вести историю изменения данных, и в случае чего, вернуться к данным до изменения.

Триггерная функция и сам триггер:

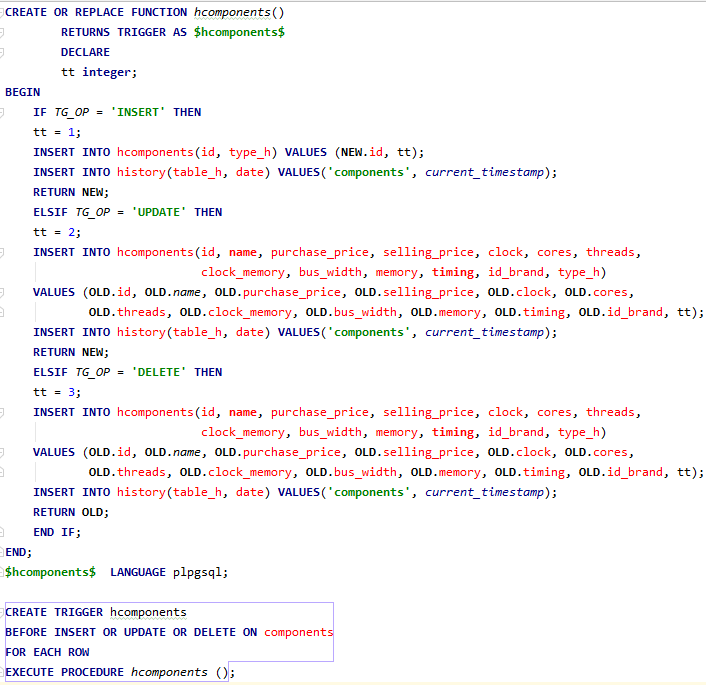


Рисунок 3 – Триггер и триггерная функция hcomponents

Итак, в среде СУБД PostgreSQL были созданы все необходимые таблицы БД магазина компьютерных комплектующих, определены первичные ключи, проведены связи между таблицами. Также таблицы были заполнены всеми необходимыми данными для работы с базой данных. Были реализованы триггеры и триггерные функции для обеспечения поддержки темпоральности данных.

3 Разработка клиентского интерфейса для работы с базой данных

Интерфейс для взаимодействия пользователей с базой данных должен соответствовать некоторым требованиям, а именно: быть дружественным, интуитивно понятным и позволять каждому пользователю работать с базой в соответствии с его полномочиями. То есть, должны быть предусмотрены различные роли входа, для каждой роли должны быть реализованы свои функции. Так, например, клиент может только добавлять товары в корзину, подтверждать заказ и иметь доступ к просмотру списка товаров и их остатку, но не может изменять характеристики или остаток товара на складе, как кладовщик. А кладовщик не имеет доступа к корзине и списку клиентов, как менеджер, и т. д.

Кроме того, интерфейс должен предусматривать возможность взаимодействия с СУБД. Инеобходимо предусмотреть поддержку резервного копирования данных и реализовать возможность выгрузки БД в файл и загрузки из него прямо в клиентском интерфейсе.

3.1 Выбор среды для разработки приложения

Практически все современные языки программирования имеют возможность подключения и работы с СУБД. Поэтому следует определиться с выбором языка, рассмотри несколько из них, а именно: С++, Python и С#.

C++ — компилируемый, статически типизированный язык программирования общего назначения. Поддерживает такие парадигмы программирования, как процедурное программирование, объектно-ориентированное программирование, обобщённое программирование. Язык имеет богатую стандартную библиотеку, которая включает в себя распространённые контейнеры и алгоритмы, ввод-вывод, регулярные выражения, поддержку многопоточности и другие возможности. C++ сочетает свойства как высокоуровневых, так и низкоуровневых языков.

С# имеет статическую типизацию, поддерживает полиморфизм, перегрузку операторов (в том числе операторов явного и неявного приведения типа), делегаты, атрибуты, события, свойства, обобщённые типы и методы, итераторы, анонимные функции с поддержкой замыканий, LINQ, исключения, комментарии в формате XML.

Python — высокоуровневый язык программирования общего назначения, ориентированный на повышение производительности разработчика и читаемости кода. Синтаксис ядра Python минималистичен. В то же время стандартная библиотека включает большой объём полезных функций. Python поддерживает структурное, объектно-ориентированное, функциональное, императивное и аспектно-ориентированное программирование. Основные архитектурные черты — динамическая типизация, автоматическое управление памятью, полная интроспекция, механизм обработки исключений, поддержка многопоточных вычислений, высокоуровневые структуры данных. Поддерживается разбиение программ на модули, которые, в свою очередь, могут объединяться в пакеты.

Для удобной реализации перечисленных требований к интерфейсу БД магазина компьютерных комплектующих был выбран язык Python. Python поддерживает практически все распространенные операционные системы. Он может прекрасно работать на карманных компьютерах, так и на больших серверах. В случае, если платформа значительно устаревает, она исключается из поддержки ядра. К примеру, версии языка, начиная от 2.6, уже не работают с платформами Windows 95, 98 и ME. В случае необходимости можно воспользоваться более старыми версиями, отказавшись от применения современных инструментов языка. И тогда приложение будет работать в том числе с этими ОС. Для старых версий периодически выходят обновления. Язык также может поддерживать работу с виртуальной машиной Java. [4].

Язык имеет четко структурированное семантическое ядро и достаточно простой синтаксис. Все, что пишется на этом языке, всегда легко читаемо. В случае необходимости передать аргументы язык использует функцию call-by-sharing.

Набор операторов в языке вполне стандартен. Удобная особенность синтаксиса – это форматирование текста кода при помощи разбивки их на блоки с помощью отступов, которые создают нажатием клавиш «Space» и «Tab». В синтаксисе отсутствуют фигурные или операторные скобки, обозначающие начало и конец блока. Такое решение заметно сокращает количество строк тела программы и приучает программиста соблюдать хороший стиль и аккуратность при написании кода [4].

Существует множество сред разработки, подходящих для решения поставленной задачи, например, PyCharm, MS VisualStudio Code, Atom.

Atom достаточно легкий бесплатный редактор, в котором просто разобраться новичку, но в то же время в нём нельзя пролистать ниже последней строки, и список всех ошибок в коде выводится в отдельном окне, что неудобно при отладке.

MS VisualStudio Code поддерживает полный цикл разработки, имеет четкие вертикальные отступы и встроенную поддержку Git, но он не имеет поддержку операционной системы Linux, вследствие чего данный редактор также не подходит для разработки клиентского интерфейса.

Наконец, PyCharm. Данная среда является наиболее подходящей для решения поставленной задачи, так как является бесплатной, официальной, обладает большим функционалом, имеет широкие возможности по разработке приложений. Поэтому интерфейс для взаимодействия пользователей с базой данных был реализован на языке Python в программной среде PyCharm.

Наиболее простой и удобной платформой для разработки веб-приложений является остается Django.

Django — бесплатный и свободный фреймворк для веб-приложений, написанный на Python. Фреймворк — это набор компонентов, которые помогают разрабатывать веб-сайты быстро и просто.

Каждый раз при разработке веб-сайтов требуются похожие компоненты: способ аутентифицировать пользователей (вход, выход, регистрация), панель управления сайтом, формы, инструменты для загрузки файлов и т. д.

К счастью для нас, другие люди обратили внимание на возникновение однотипных проблем при веб-разработке, так что они объединились и создали фреймворки (Django и другие), которые предлагают нам готовые шаблоны для использования.

Фреймворки существуют, чтобы облегчить процесс разработки и позволить нам не изобретать колесо.

3.2 Описание основных классов и методов интерфейса

Для реализации клиентского интерфейса были реализованы следующие основные классы: AuthView, MainView, DB, где AuthView и MainView отвечают за интерфейсы авторизации и работы с базой данных, а класс DB отвечает за запись или чтение из базы данных.

Общий класс AuthView отвечает за диалог с пользователем при запуске программы; в классе реализован метод Index, совершающий попытку подключения по введенным данным (логин и пароль пользователя). В случае удачного подключения открывается новое окно для работы с базой данных магазина компьютерных комплектующих.

Класс MainView содержит следующие методы:

1. Show\_tables – метод выводит список всех таблиц, существующих в базе данных для дальнейшего доступа к каждой из них, вызывается при открытии окна;

2. Show\_table – метод выводит список всех записей из выбранной таблицы;

3. Show\_table\_old – метод выводит список всех записей, к которым можно вернуться из выбранной таблицы;

4. Enter\_table – метод вызывается при нажатии кнопки «Add», записывает в базу данных введённую информацию для выбранного поля и таблицы;

5. Back\_num – метод делает выбранное пользователем количество откатов записей в выбранной таблице.

Класс DB cодержит следующие методы:

1. Get\_tables – метод возвращает список всех таблиц, созданных в базе данных;

2. Output\_titles – метод возвращает список всех названий полей из выбранной таблицы;

3. Output\_tables – метод возвращает массив всех записей в выбранную таблицу;

4. Update – метод записывает в выбранную таблицу новые значения для указанной записи;

5. Del\_string – метод удаляет из выбранной таблицы указанную запись;

6. Insert – метод добавляет в указанную таблицу новую запись с введёнными значениями для каждого поля;

7. Search\_component – метод возвращает найденные в базе данных значения, которые указал пользователь;

8. Back\_num – метод делает выбранное пользователем количество откатов записей в выбранной таблице;

9. Backup – метод создает резервную копию базы данных;

10. Recovery – метод делает откат к выбранной резервной копии базы данных;

UML-диаграмма классов находится в Приложении В.

Исходный код приложения находится в Приложении Д.

3.3 Реализация создания резервной копии и восстановления базы данных

При работе с базой данной существует необходимость создания резервной копии или выгрузки базы данных в файл SQL-формата и восстановления. Данная процедура реализована с помощью стандартных средст СУБД PostgreSQL, а именно используются утилиты pg\_dump – позволяет выгружать содержимое, вместе с сущесвующими таблицами, тригерами и тригерными функциями в файл в виде SQL кода; и утилита psql – позволяет восстанавливать базу данных, используя ранее созданный файл резервной копии.

Данная процедура выполняется из клиентского интерфейса, и реализована двумя основными методами класса «DB»:

Backup – данный метод вызывается при нажатии кнопки «Create backup», выполняет последовательность команд по вызову утилиты pg\_dump и сохраняет полученный файл в каталоге с программой.

Recovery – данный метод вызывается при нажатии кнопки «Recovery» после того, как пользователь выбирает файл восстановления, выполняет последовательность команды по вызову утилиты psql.

Таким образом существует возможность прямо из клиентского интерфейса создавать резервную копию (или архивировать для переноса) базы данных. И так же восстанавливать в случае необходимости.

3.4 Пример работы клиентского интерфейса

Запуск программы, диалоговое окно входа в учетную запись:

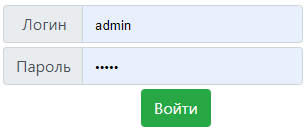


Рисунок 4 – Авторизация пользователя

Окно для работы с БД после успешной идентификации менеджера:

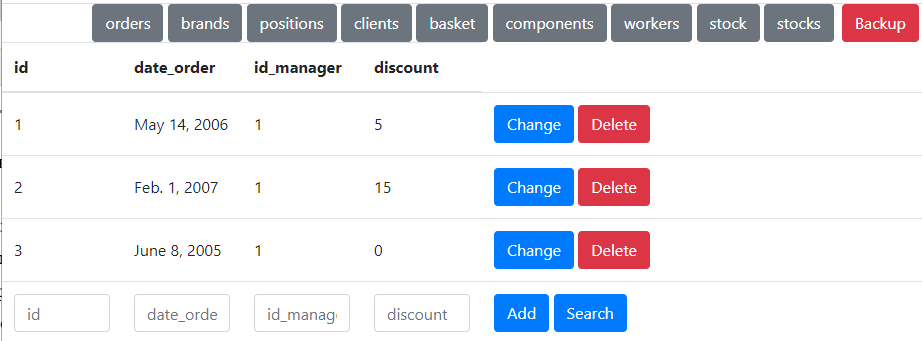


Рисунок 5 – Окно для работы с БД для менеджера

Окно для работы с БД после успешной идентификации клиента:

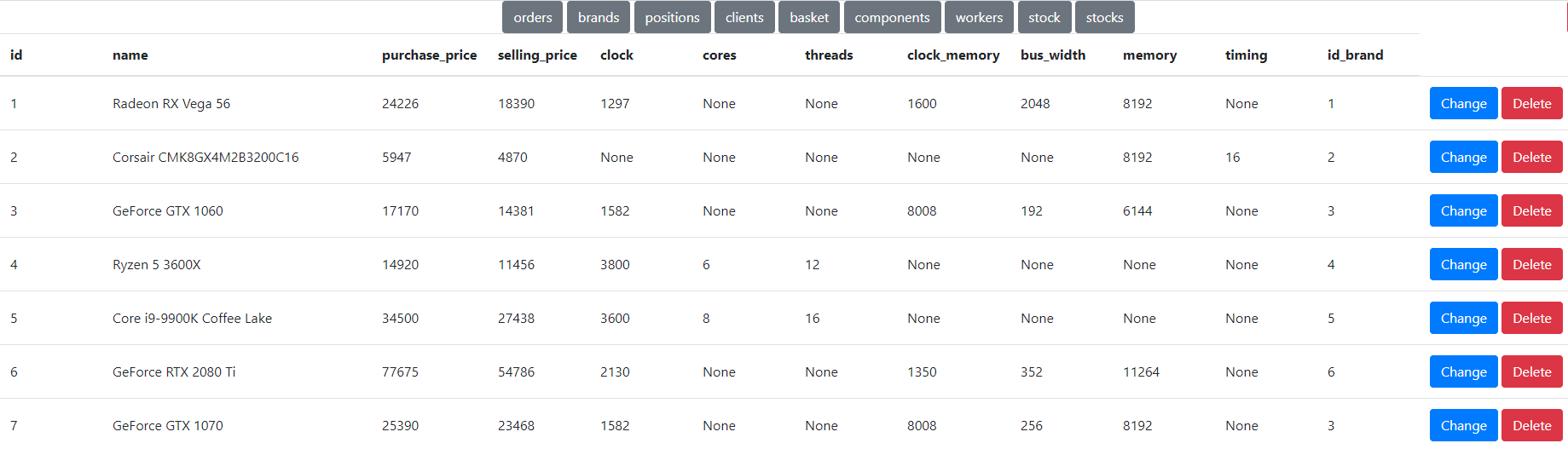


Рисунок 6 – Окно для работы с БД для клиента

Окно для работы с БД после успешной идентификации кладовщика:

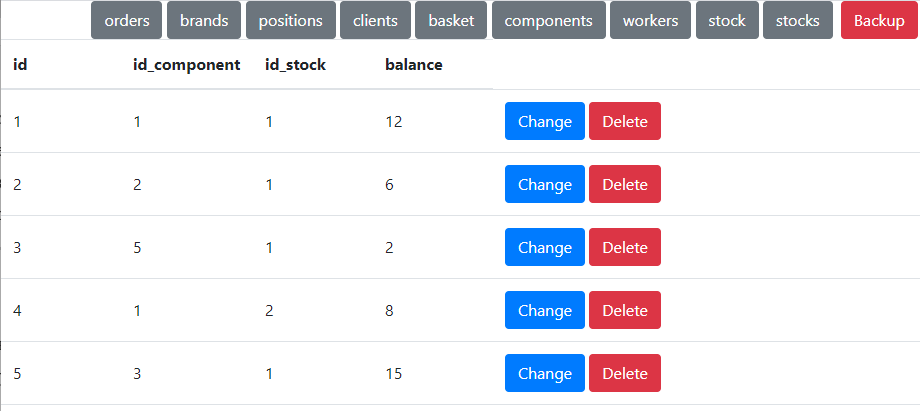


Рисунок 7 – Окно для работы с БД для кладовщика

Поиск по значению:

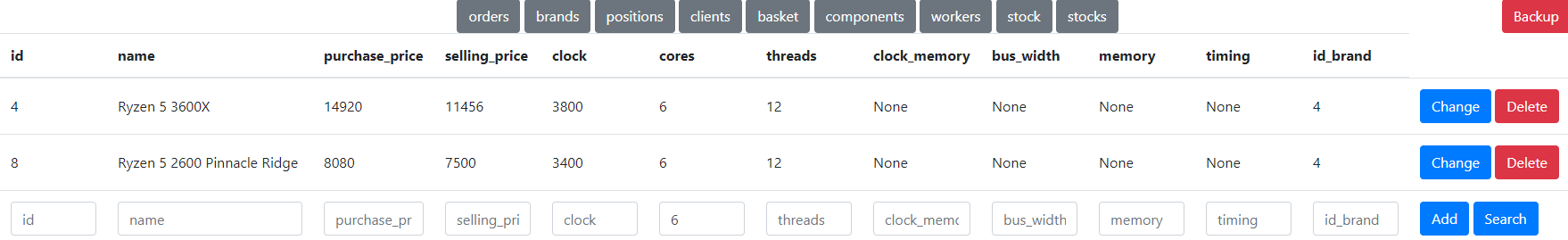


Рисунок 8 – Поиск по значению

Создание резервной копии и восстановления базы данных:

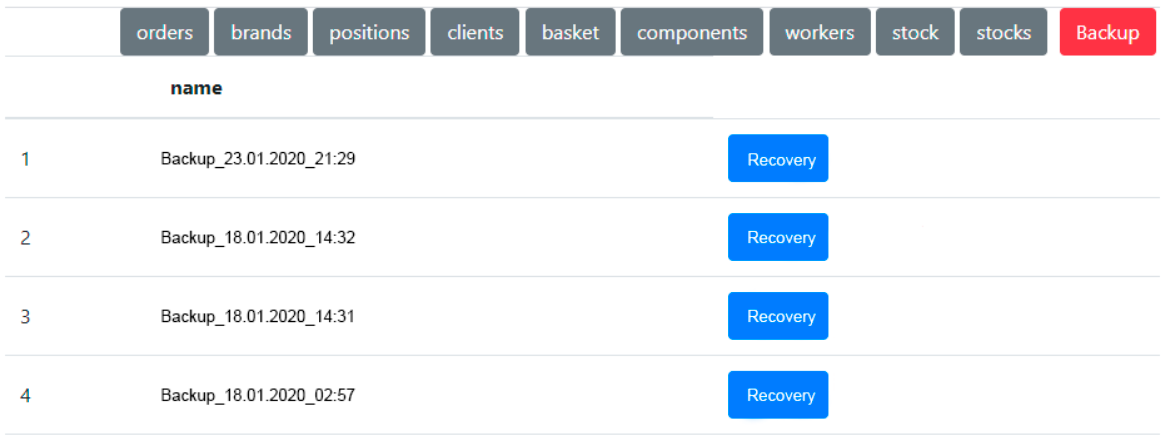


Рисунок 9 – Резервное копирование и восстановление

Итак, в рамках поставленной задачи было реализовано приложение для удобной и корректной работы с базой данных магазина компьютерных комплектующих с дружественным интерфейсом. Были выполнены все поставленные к интерфейсу требования.

ЗАКЛЮЧЕНИЕ

В рамках данной курсовой работы была поставлена задача проектирования и реализации базы данных магазина компьютерных комплектующих. Помимо эффективной реализации базы данных, требовалось написать программное средство с интуитивно понятным интерфейсом, а также разграничить права доступа в связи с обязанностями конечного пользователя для работы с базой данных.

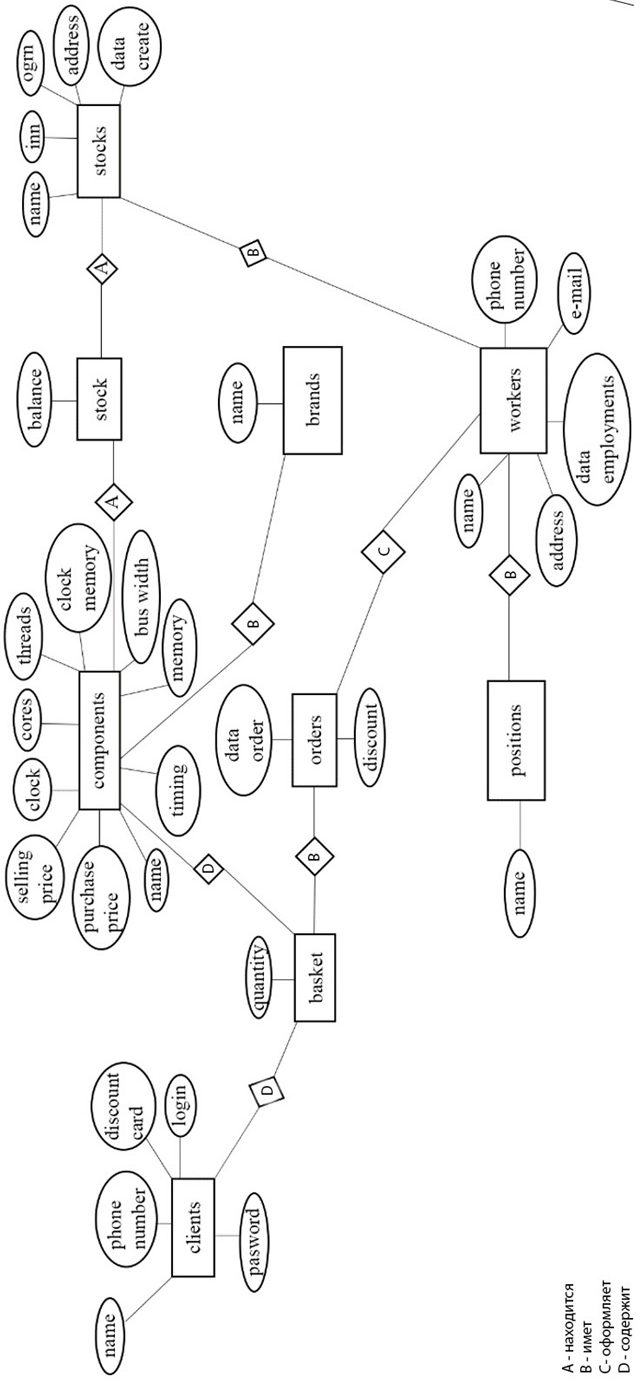
Для решения поставленной задачи была изучена предметная область, а именно устройство и работа магазина компьютерных комплектующих, определены конечные пользователи базы данных, описаны объекты и их атрибуты. Также была разработана концептуальная модель в виде ER-диаграммы. Составлена даталогическая схема отношений. После с помощью СУБД PostgreSQL была реализована физическая модель базы данных магазина компьютерных комплектующих. И на языке Python был написан клиентский интерфейс для корректной работы с данной БД.

Данную реализацию базы данных можно модернизировать с дальнейшими пожеланиями и требованиями конечных пользователей, а также с изменениями в работе магазина компьютерных комплектующих. Чему способствует выбранная СУБД и язык реализации клиентского интерфейса.

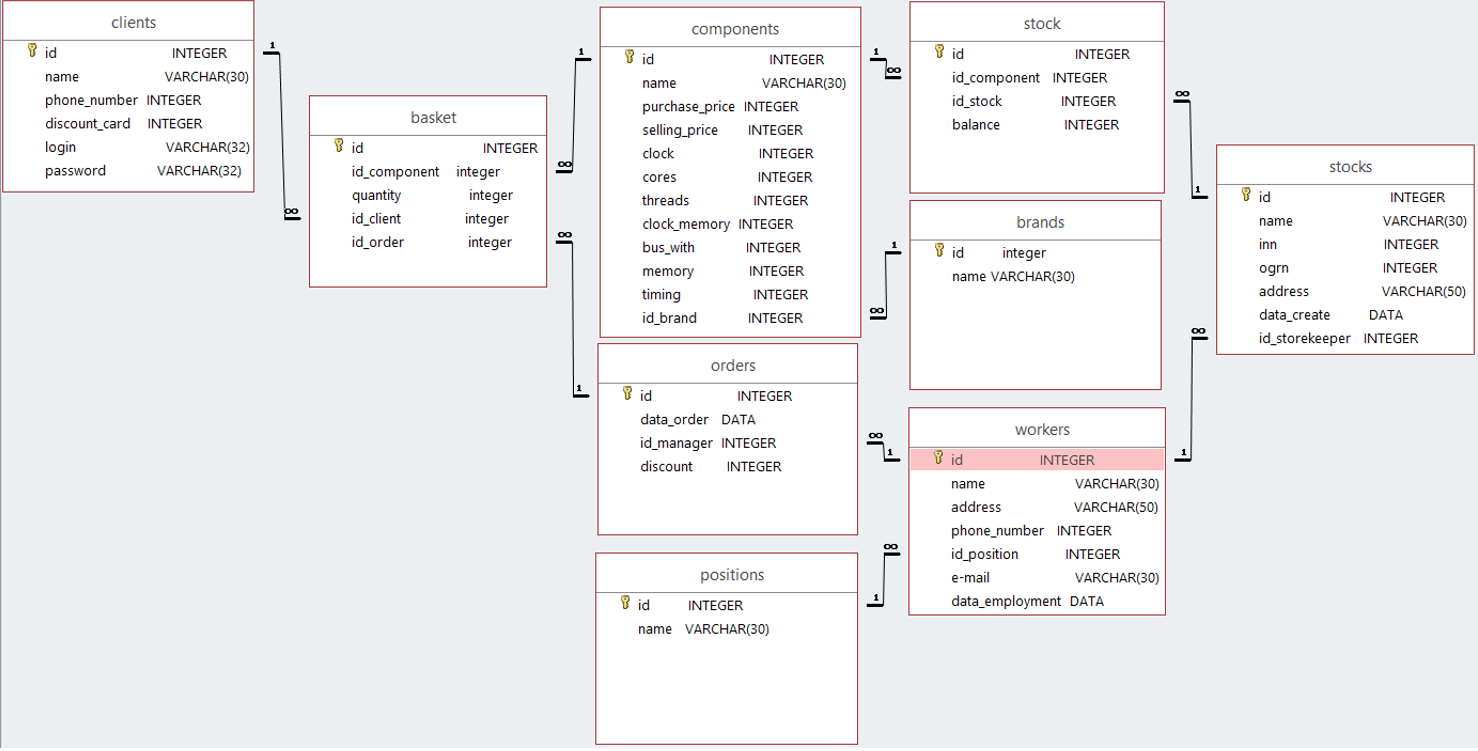
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Дейт К.Дж. Введение в системы баз данных, 6-е издание: Пер. с англ. – К.; М.; СПб.: Издательский дом «Вильямс», 2000. – 848 с.
2. Конноли Т., Бегг К., Страчан А. Базы данных: проектирование, реализация и сопровождение. Теория и практика, 2-е изд.: Пер. с англ. – М.: Издательский дом «Вильямс», 2000. – 1120 с.
3. Карпова Т.С. Базы данных: модели, разработка, реализация. – СПб.: Питер, 2001. – 304 с.
4. М. Лутц. Изучаем Python. 4-е изд.: – СПб.: Питер, 2019. – 504 с.
5. [Паттон Джефф](https://www.ozon.ru/person/patton-dzheff-70734305/). Пользовательские истории. Искусство гибкой разработки ПО. – СПб.: Питер, 2019. – 288 с.

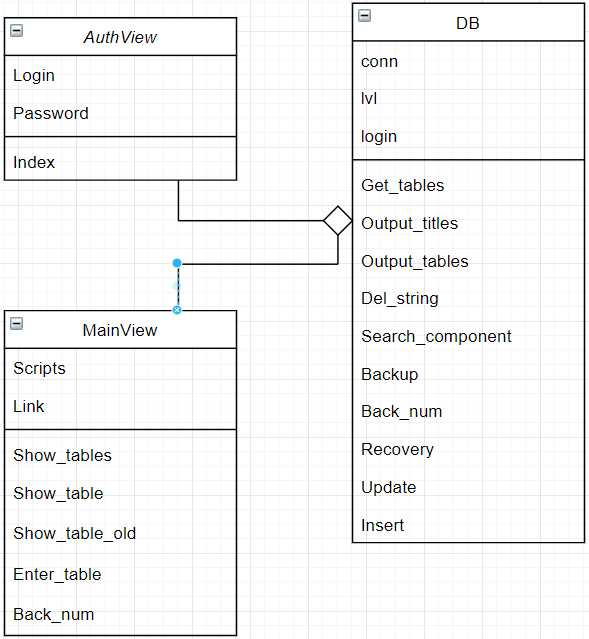
ПРИЛОЖЕНИЕ А



ПРИЛОЖЕНИЕ Б



ПРИЛОЖЕНИЕ В



ПРИЛОЖЕНИЕ Г

CREATE DATABASE PC\_shop

WITH

OWNER = user\_1

ENCODING = 'UTF8'

LC\_COLLATE = 'Russian\_Russia.1251'

LC\_CTYPE = 'Russian\_Russia.1251'

TABLESPACE = pg\_default

CONNECTION LIMIT = -1;

CREATE TABLE IF NOT EXISTS brands(

id SERIAL PRIMARY KEY,

name VARCHAR (30)

);

CREATE TABLE IF NOT EXISTS components(

id SERIAL PRIMARY KEY,

name VARCHAR(30),

purchase\_price INTEGER,

selling\_price INTEGER,

clock INTEGER,

cores INTEGER,

threads INTEGER,

clock\_memory INTEGER,

bus\_width INTEGER,

memory INTEGER,

timing INTEGER,

id\_brand INTEGER REFERENCES brands(id) ON DELETE CASCADE ON UPDATE CASCADE

);

CREATE TABLE IF NOT EXISTS clients(

id SERIAL PRIMARY KEY,

name VARCHAR(30),

phone\_number VARCHAR(11),

discount\_card VARCHAR(9),

login VARCHAR(32) UNIQUE ,

password VARCHAR(32)

);

CREATE TABLE IF NOT EXISTS positions(

id SERIAL PRIMARY KEY,

name VARCHAR(30)

);

CREATE TABLE IF NOT EXISTS workers(

id SERIAL PRIMARY KEY,

login VARCHAR(32) UNIQUE ,

password VARCHAR(32),

name VARCHAR(30),

address VARCHAR(50),

phone\_number VARCHAR(11),

id\_position INTEGER REFERENCES positions(id) ON DELETE CASCADE ON UPDATE CASCADE,

email VARCHAR(30),

date\_employment DATE

);

CREATE TABLE IF NOT EXISTS stocks(

id SERIAL PRIMARY KEY,

name VARCHAR(30),

inn INTEGER,

ogrn INTEGER,

address VARCHAR(50),

date\_create DATE,

id\_storekeeper INTEGER REFERENCES workers(id) ON DELETE CASCADE ON UPDATE CASCADE

);

CREATE TABLE IF NOT EXISTS stock(

id SERIAL PRIMARY KEY,

id\_component INTEGER REFERENCES components(id) ON DELETE CASCADE ON UPDATE CASCADE,

id\_stock INTEGER REFERENCES stocks(id) ON DELETE CASCADE ON UPDATE CASCADE,

balance INTEGER

);

CREATE TABLE IF NOT EXISTS orders(

id SERIAL PRIMARY KEY,

date\_order DATE,

id\_manager INTEGER REFERENCES workers(id) ON DELETE CASCADE ON UPDATE CASCADE,

discount INTEGER

);

CREATE TABLE IF NOT EXISTS basket(

id SERIAL PRIMARY KEY,

id\_component INTEGER REFERENCES components(id) ON DELETE CASCADE ON UPDATE CASCADE,

quantity INTEGER,

id\_client INTEGER REFERENCES clients(id) ON DELETE CASCADE ON UPDATE CASCADE,

id\_order INTEGER REFERENCES orders(id) ON DELETE CASCADE ON UPDATE CASCADE

);

CREATE OR REPLACE FUNCTION hcomponents()

RETURNS TRIGGER AS $hcomponents$

DECLARE

tt integer;

BEGIN

IF TG\_OP = 'INSERT' THEN

tt = 1;

INSERT INTO hcomponents(id, type\_h) VALUES (NEW.id, tt);

INSERT INTO history(table\_h, date) VALUES('components', current\_timestamp);

RETURN NEW;

ELSIF TG\_OP = 'UPDATE' THEN

tt = 2;

INSERT INTO hcomponents(id, name, purchase\_price, selling\_price, clock, cores, threads, clock\_memory, bus\_width, memory, timing, id\_brand, type\_h)

VALUES (OLD.id, OLD.name, OLD.purchase\_price, OLD.selling\_price, OLD.clock, OLD.cores, OLD.threads, OLD.clock\_memory, OLD.bus\_width, OLD.memory, OLD.timing, OLD.id\_brand, tt);

INSERT INTO history(table\_h, date) VALUES('components', current\_timestamp);

RETURN NEW;

ELSIF TG\_OP = 'DELETE' THEN

tt = 3;

INSERT INTO hcomponents(id, name, purchase\_price, selling\_price, clock, cores, threads, clock\_memory, bus\_width, memory, timing, id\_brand, type\_h)

VALUES (OLD.id, OLD.name, OLD.purchase\_price, OLD.selling\_price, OLD.clock, OLD.cores, OLD.threads, OLD.clock\_memory, OLD.bus\_width, OLD.memory, OLD.timing, OLD.id\_brand, tt);

INSERT INTO history(table\_h, date) VALUES('components', current\_timestamp);

RETURN OLD;

END IF;

END;

$hcomponents$ LANGUAGE plpgsql;

CREATE TRIGGER hcomponents

BEFORE INSERT OR UPDATE OR DELETE ON components

FOR EACH ROW

EXECUTE PROCEDURE hcomponents ();

CREATE OR REPLACE FUNCTION hclients()

RETURNS TRIGGER AS $hclients$

DECLARE

tt integer;

BEGIN

IF TG\_OP = 'INSERT' THEN

tt = 1;

INSERT INTO hclients(id, type\_h) VALUES (NEW.id, tt);

INSERT INTO history(table\_h, date) VALUES('clients', current\_timestamp);

RETURN NEW;

ELSIF TG\_OP = 'UPDATE' THEN

tt = 2;

INSERT INTO hclients(id, name, phone\_number, discount\_card, login, password, type\_h)

VALUES (OLD.id, OLD.name, OLD.phone\_number, OLD.discount\_card, OLD.login, OLD.password, tt);

INSERT INTO history(table\_h, date) VALUES('clients', current\_timestamp);

RETURN NEW;

ELSIF TG\_OP = 'DELETE' THEN

tt = 3;

INSERT INTO hclients(id, name, phone\_number, discount\_card, login, password, type\_h)

VALUES (OLD.id, OLD.name, OLD.phone\_number, OLD.discount\_card, OLD.login, OLD.password, tt);

INSERT INTO history(table\_h, date) VALUES('clients', current\_timestamp);

RETURN OLD;

END IF;

END;

$hclients$ LANGUAGE plpgsql;

CREATE TRIGGER hclients

BEFORE INSERT OR UPDATE OR DELETE ON clients

FOR EACH ROW

EXECUTE PROCEDURE hclients ();

CREATE OR REPLACE FUNCTION hbasket()

RETURNS TRIGGER AS $hbasket$

DECLARE

tt integer;

BEGIN

IF TG\_OP = 'INSERT' THEN

tt = 1;

INSERT INTO hbasket(id, type\_h) VALUES (NEW.id, tt);

INSERT INTO history(table\_h, date) VALUES('basket', current\_timestamp);

RETURN NEW;

ELSIF TG\_OP = 'UPDATE' THEN

tt = 2;

INSERT INTO hbasket(id, id\_component, quantity, id\_client, id\_order, type\_h)

VALUES (OLD.id, OLD.id\_component, OLD.quantity, OLD.id\_client, OLD.id\_order, tt);

INSERT INTO history(table\_h, date) VALUES('basket', current\_timestamp);

RETURN NEW;

ELSIF TG\_OP = 'DELETE' THEN

tt = 3;

INSERT INTO hbasket(id, id\_component, quantity, id\_client, id\_order, type\_h)

VALUES (OLD.id, OLD.id\_component, OLD.quantity, OLD.id\_client, OLD.id\_order, tt);

INSERT INTO history(table\_h, date) VALUES('basket', current\_timestamp);

RETURN OLD;

END IF;

END;

$hbasket$ LANGUAGE plpgsql;

CREATE TRIGGER hbasket

BEFORE INSERT OR UPDATE OR DELETE ON basket

FOR EACH ROW

EXECUTE PROCEDURE hbasket ();

CREATE OR REPLACE FUNCTION horders()

RETURNS TRIGGER AS $horders$

DECLARE

tt integer;

BEGIN

IF TG\_OP = 'INSERT' THEN

tt = 1;

INSERT INTO horders(id, type\_h) VALUES (NEW.id, tt);

INSERT INTO history(table\_h, date) VALUES('orders', current\_timestamp);

RETURN NEW;

ELSIF TG\_OP = 'UPDATE' THEN

tt = 2;

INSERT INTO horders(id, data\_order, id\_manager, discount, type\_h)

VALUES (OLD.id, OLD.data\_order, OLD.id\_manager, OLD.discount, tt);

INSERT INTO history(table\_h, date) VALUES('orders', current\_timestamp);

RETURN NEW;

ELSIF TG\_OP = 'DELETE' THEN

tt = 3;

INSERT INTO horders(id, data\_order, id\_manager, discount, type\_h)

VALUES (OLD.id, OLD.data\_order, OLD.id\_manager, OLD.discount, tt);

INSERT INTO history(table\_h, date) VALUES('orders', current\_timestamp);

RETURN OLD;

END IF;

END;

$horders$ LANGUAGE plpgsql;

CREATE TRIGGER horders

BEFORE INSERT OR UPDATE OR DELETE ON orders

FOR EACH ROW

EXECUTE PROCEDURE horders ();

CREATE OR REPLACE FUNCTION hpositions()

RETURNS TRIGGER AS $hpositions$

DECLARE

tt integer;

BEGIN

IF TG\_OP = 'INSERT' THEN

tt = 1;

INSERT INTO hpositions(id, type\_h) VALUES (NEW.id, tt);

INSERT INTO history(table\_h, date) VALUES('positions', current\_timestamp);

RETURN NEW;

ELSIF TG\_OP = 'UPDATE' THEN

tt = 2;

INSERT INTO hpositions(id, name, type\_h)

VALUES (OLD.id, OLD.name, tt);

INSERT INTO history(table\_h, date) VALUES('positions', current\_timestamp);

RETURN NEW;

ELSIF TG\_OP = 'DELETE' THEN

tt = 3;

INSERT INTO hpositions(id, name, type\_h)

VALUES (OLD.id, OLD.name, tt);

INSERT INTO history(table\_h, date) VALUES('positions', current\_timestamp);

RETURN OLD;

END IF;

END;

$hpositions$ LANGUAGE plpgsql;

CREATE TRIGGER hpositions

BEFORE INSERT OR UPDATE OR DELETE ON positions

FOR EACH ROW

EXECUTE PROCEDURE hpositions ();

CREATE OR REPLACE FUNCTION hstock()

RETURNS TRIGGER AS $hstock$

DECLARE

tt integer;

BEGIN

IF TG\_OP = 'INSERT' THEN

tt = 1;

INSERT INTO hstock(id, type\_h) VALUES (NEW.id, tt);

INSERT INTO history(table\_h, date) VALUES('stock', current\_timestamp);

RETURN NEW;

ELSIF TG\_OP = 'UPDATE' THEN

tt = 2;

INSERT INTO hstock(id, id\_component, id\_stock, balance, type\_h)

VALUES (OLD.id, OLD.id\_component, OLD.id\_stock, OLD.balance, tt);

INSERT INTO history(table\_h, date) VALUES('stock', current\_timestamp);

RETURN NEW;

ELSIF TG\_OP = 'DELETE' THEN

tt = 3;

INSERT INTO hstock(id, id\_component, id\_stock, balance, type\_h)

VALUES (OLD.id, OLD.id\_component, OLD.id\_stock, OLD.balance, tt);

INSERT INTO history(table\_h, date) VALUES('stock', current\_timestamp);

RETURN OLD;

END IF;

END;

$hstock$ LANGUAGE plpgsql;

CREATE TRIGGER hstock

BEFORE INSERT OR UPDATE OR DELETE ON stock

FOR EACH ROW

EXECUTE PROCEDURE hstock ();

CREATE OR REPLACE FUNCTION hbrands()

RETURNS TRIGGER AS $hbrands$

DECLARE

tt integer;

BEGIN

IF TG\_OP = 'INSERT' THEN

tt = 1;

INSERT INTO hbrands(id, type\_h) VALUES (NEW.id, tt);

INSERT INTO history(table\_h, date) VALUES('brands', current\_timestamp);

RETURN NEW;

ELSIF TG\_OP = 'UPDATE' THEN

tt = 2;

INSERT INTO hbrands(id, name, type\_h)

VALUES (OLD.id, OLD.name, tt);

INSERT INTO history(table\_h, date) VALUES('brands', current\_timestamp);

RETURN NEW;

ELSIF TG\_OP = 'DELETE' THEN

tt = 3;

INSERT INTO hbrands(id, name, type\_h)

VALUES (OLD.id, OLD.name, tt);

INSERT INTO history(table\_h, date) VALUES('brands', current\_timestamp);

RETURN OLD;

END IF;

END;

$hbrands$ LANGUAGE plpgsql;

CREATE TRIGGER hbrands

BEFORE INSERT OR UPDATE OR DELETE ON brands

FOR EACH ROW

EXECUTE PROCEDURE hbrands ();

CREATE OR REPLACE FUNCTION hworkers()

RETURNS TRIGGER AS $hworkers$

DECLARE

tt integer;

BEGIN

IF TG\_OP = 'INSERT' THEN

tt = 1;

INSERT INTO hworkers(id, type\_h) VALUES (NEW.id, tt);

INSERT INTO history(table\_h, date) VALUES('workers', current\_timestamp);

RETURN NEW;

ELSIF TG\_OP = 'UPDATE' THEN

tt = 2;

INSERT INTO hworkers(id, name, address, phone\_number, id\_position, e-mail, data\_employment, type\_h)

VALUES (OLD.id, OLD.name, OLD.address, OLD.phone\_number, OLD.id\_position, OLD.e-mail, OLD.data\_employment, tt);

INSERT INTO history(table\_h, date) VALUES('workers', current\_timestamp);

RETURN NEW;

ELSIF TG\_OP = 'DELETE' THEN

tt = 3;

INSERT INTO hworkers(id, name, address, phone\_number, id\_position, e-mail, data\_employment, type\_h)

VALUES (OLD.id, OLD.name, OLD.address, OLD.phone\_number, OLD.id\_position, OLD.e-mail, OLD.data\_employment, tt);

INSERT INTO history(table\_h, date) VALUES('workers', current\_timestamp);

RETURN OLD;

END IF;

END;

$hworkers$ LANGUAGE plpgsql;

CREATE TRIGGER hworkers

BEFORE INSERT OR UPDATE OR DELETE ON workers

FOR EACH ROW

EXECUTE PROCEDURE hworkers ();

CREATE OR REPLACE FUNCTION hstocks()

RETURNS TRIGGER AS $hstocks$

DECLARE

tt integer;

BEGIN

IF TG\_OP = 'INSERT' THEN

tt = 1;

INSERT INTO hstocks(id, type\_h) VALUES (NEW.id, tt);

INSERT INTO history(table\_h, date) VALUES('stocks', current\_timestamp);

RETURN NEW;

ELSIF TG\_OP = 'UPDATE' THEN

tt = 2;

INSERT INTO hstocks(id, name, inn, ogrn, address, data\_create, id\_storekeeper, type\_h)

VALUES (OLD.id, OLD.name, OLD.inn, OLD.ogrn, OLD.address, OLD.data\_create, OLD.id\_storekeeper, tt);

INSERT INTO history(table\_h, date) VALUES('stocks', current\_timestamp);

RETURN NEW;

ELSIF TG\_OP = 'DELETE' THEN

tt = 3;

INSERT INTO hstocks(id, name, inn, ogrn, address, data\_create, id\_storekeeper, type\_h)

VALUES (OLD.id, OLD.name, OLD.inn, OLD.ogrn, OLD.address, OLD.data\_create, OLD.id\_storekeeper, tt);

INSERT INTO history(table\_h, date) VALUES('stocks', current\_timestamp);

RETURN OLD;

END IF;

END;

$hstocks$ LANGUAGE plpgsql;

CREATE TRIGGER hstocks

BEFORE INSERT OR UPDATE OR DELETE ON stocks

FOR EACH ROW

EXECUTE PROCEDURE hstocks ();

ПРИЛОЖЕНИЕ Д

import psycopg2

from psycopg2 import sql

class DB():

def \_\_init\_\_(self, login, password):

self.conn = psycopg2.connect(dbname='dima\_lab5', user='user\_1',

password='password', host='62.109.15.226')

self.login = None

self.lvl = None

with self.conn.cursor() as cursor:

columns = ['id', 'login', 'password']

stmt = sql.SQL('SELECT {} FROM {}').format(

sql.SQL(',').join(map(sql.Identifier, columns)),

sql.Identifier('clients')

)

cursor.execute(stmt)

for row in cursor:

if login.lower() == row[1] and password == row[2]:

self.login = login.lower()

self.lvl = 0

break

if not self.login:

columns = ['id', 'login', 'password', 'id\_position']

stmt = sql.SQL('SELECT {} FROM {}').format(

sql.SQL(',').join(map(sql.Identifier, columns)),

sql.Identifier('workers')

)

cursor.execute(stmt)

for row in cursor:

if login.lower() == row[1] and password == row[2]:

self.login = login

self.lvl = row[3]

if not self.login:

self.conn = None

def get\_tables(self):

with self.conn.cursor() as cursor:

stmt = sql.SQL("SELECT table\_name FROM information\_schema.tables"

" WHERE table\_schema NOT IN ('information\_schema','pg\_catalog') AND "

"table\_name NOT LIKE '%\_old';")

cursor.execute(stmt)

return self.beautiful\_change(cursor.fetchall())

def output\_titles(self, table):

with self.conn.cursor() as cursor:

stmt = sql.SQL("SELECT column\_name FROM information\_schema.columns WHERE table\_name = '{}';".format(table))

cursor.execute(stmt)

return self.beautiful\_change(cursor.fetchall())

def output\_tables(self, table):

with self.conn.cursor() as cursor:

stmt = sql.SQL('SELECT \* FROM {};'.format(table))

cursor.execute(stmt)

return self.con(self.output\_titles(table), cursor.fetchall())

def update(self, mass, table):

with self.conn.cursor() as cursor:

stmt = self.crt\_update(mass, table)

cursor.execute(stmt)

def del\_string(self, old, table):

with self.conn.cursor() as cursor:

stmt = "DELETE FROM {} WHERE id = {} ;".format(table, old)

cursor.execute(stmt)

def insert(self, mass, table):

with self.conn.cursor() as cursor:

stmt = self.crt\_insert(mass, table)

print(stmt)

cursor.execute(stmt)

def search\_component(self, table, args):

prov = False

for i in args:

if a != '':

prov = True

with self.conn.cursor() as cursor:

stmt = sql.SQL('SELECT \* FROM {} WHERE {};'.format(table,

self.create\_nice\_search(args=args,

titles=self.output\_titles(table)

)))

if prov:

cursor.execute(stmt)

return self.con(self.output\_titles(table), cursor.fetchall())

else:

return self.output\_tables(table)

def back\_num(self, num, table):

with self.conn.cursor() as cursor:

args = "SELECT COLUMN\_NAME FROM INFORMATION\_SCHEMA.COLUMNS WHERE TABLE\_NAME='{}\_old' AND COLUMN\_NAME NOT IN ('date\_operation')".format(table)

cursor.execute(args)

args = self.beautiful\_change(cursor.fetchall())

string = "{}".format(args[0])

for i in args[1:]:

string += ', ' + i

stmt = "SELECT {} FROM {}\_old ORDER BY date\_operation DESC LIMIT {};".format(string, table, num)

cursor.execute(stmt)

for i in cursor.fetchall():

if i[-1] == 'INSERT':

self.del\_string(i[0], table)

elif i[-1] == 'DELETE':

mass = []

for j in i[:-1]:

mass.append(str(j))

self.insert(mass, table)

elif i[-1] == 'UPDATE':

mass = []

for j in i[:-1]:

mass.append(str(j))

self.update(mass + [i[0]], table)

print(cursor.fetchall())

from django.shortcuts import render

from django.http import Http404, HttpResponseRedirect

from django.urls import reverse

from django.db import transaction

from lab5\_3.apps import DB

def show\_tables(request):

if DB.a.login and DB.a.lvl == 3:

return render(request, 'root/main\_root.html', {'tables': DB.a.get\_tables()})

DB.scripts = 'wrong password or login'

return HttpResponseRedirect(reverse('main\_form:index'))

def show\_components(request):

if DB.a.login and DB.a.lvl == 3:

save = DB.scripts

DB.scripts = None

link = 'http://127.0.0.1:8000/root/enter\_components/'

link2 = 'http://127.0.0.1:8000/root/enter\_components\_s/'

return render(request, 'root/show\_table.html',

{'tittles': DB.a.show\_components()[0], 'strings': DB.a.show\_components()[1:], 'link': link,

'link2': link2, 'last': DB.a.show\_components()[-1], 'scripts': save})

DB.scripts = 'wrong password or login'

return HttpResponseRedirect(reverse('main\_form:index'))

def show\_table\_old(request):

if DB.a.login and DB.a.lvl == 3:

save = DB.scripts

DB.scripts = None

if 'table' in request.POST:

DB.link = request.POST['table']

if 'link' in request.POST:

link = request.POST['link']

else:

link = DB.link

DB.link = None

DB.link = link

else:

return render(request, 'root/backups.html',

{'tables': DB.a.get\_tables(),

'table': link,

'titles': DB.a.output\_titles('{}\_old'.format(link)),

'strings': DB.a.output\_tables('{}\_old'.format(link)), 'scripts': save})

DB.scripts = 'wrong password or login'

return HttpResponseRedirect(reverse('main\_form:index'))

def show\_table(request):

if DB.a.login and DB.a.lvl == 3:

save = DB.scripts

DB.scripts = None

if 'table' in request.POST:

DB.link = request.POST['table']

if '\_del' in request.POST:

DB.a.del\_string(request.POST['id\_old'], request.POST['table'])

elif '\_edit' in request.POST:

mass = []

m = DB.a.output\_titles(DB.link)

for i in request.POST:

if i in m:

mass.append(request.POST[i])

mass.append(request.POST['id\_old'])

try:

DB.a.update(mass, DB.link)

except:

DB.a.rollback()

DB.scripts = 'incorrect input'

elif '\_add' in request.POST:

try:

mass = []

m = DB.a.output\_titles(DB.link)

for i in request.POST:

if i in m:

mass.append(request.POST[i])

DB.a.insert(mass, DB.link)

except:

DB.a.rollback()

DB.scripts = 'incorrect input'

if 'link' in request.POST:

link = request.POST['link']

else:

link = DB.link

DB.link = None

if DB.a.output\_tables(link):

last = DB.a.output\_tables(link)[-1]

else:

last = None

if '\_search' in request.POST:

DB.link = link

mass = []

m = DB.a.output\_titles(DB.link)

for i in request.POST:

if i in m:

mass.append(request.POST[i])

return render(request, 'root/show\_table.html', {

'tables': DB.a.get\_tables(),

'table': link,

'titles': DB.a.output\_titles(link),

'strings': DB.a.search\_component(request.POST['table'], mass),

'last': last,

'scripts': save

})

else:

DB.link = link

return render(request, 'root/show\_table.html',

{'tables': DB.a.get\_tables(),

'table': link,

'titles': DB.a.output\_titles(link),

'last': last,

'strings': DB.a.output\_tables(link), 'scripts': save})

DB.scripts = 'wrong password or login'

return HttpResponseRedirect(reverse('main\_form:index'))

def enter\_table(request):

if DB.a.login and DB.a.lvl == 3:

return HttpResponseRedirect(reverse('root:show\_table'))

DB.scripts = 'wrong password or login'

return HttpResponseRedirect(reverse('main\_form:index'))

def back\_num(request):

if DB.a.login and DB.a.lvl == 3:

DB.a.back\_num(request.POST['num'], request.POST['table'])

return HttpResponseRedirect(reverse('root:show\_table'))

DB.scripts = 'wrong password or login'

return HttpResponseRedirect(reverse('main\_form:index'))

from django.shortcuts import render

from django.http import Http404, HttpResponseRedirect

from django.urls import reverse

from lab5\_3.apps import DB

scripts = None

def index(request):

if 'login' in request.POST and 'password' in request.POST:

DB.a = DB.DB(request.POST['login'], request.POST['password'], )

if DB.a.login:

return HttpResponseRedirect(reverse('root:show\_tables'))

DB.scripts = 'wrong password or login'

return HttpResponseRedirect(reverse('main\_form:index'))

return render(request, 'authentication\_form/main\_form.html', {'scripts': DB.scripts})

ПРИЛОЖЕНИЕ Е

**Отзыв руководителя**



МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ**

**ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**

**«ДОНСКОЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»**

**(ДГТУ)**

**ОТЗЫВ**

**руководителя на курсовую работу**

Кононова Дмитрия Сергеевича (Ф.И.О. студента)

Разработка базы данных «Магазин компьютерных комплектующих»

(наименование темы КР)

представленный к защите по направлению/специальности

09.03.04 Программная инженерия (код и наименование направления специальности подготовки)

Программное обеспечение вычислительной техники и автоматизированных систем (наименования профиля/специализация)

/ Новиков С. П. /

(должность) (подпись) (Ф.И.О)

« » 2020 г.