**СОДЕРЖАНИЕ**

**ВВЕДЕНИЕ** 3

**ГЛАВА 1 РАБОТА НАД БАЗОЙ ДАННЫХ** 4

1.1 Создание макета базы данных 4

1.2 Реализация и наполнение информацией базы данных 5

1.3 Доработка функционала базы данных 6

1.4 Запросы отчётов 8

**ГЛАВА 2 РАБОТА НАД БАЗОЙ ДАННЫХ** 9

2.1 Преднастройка рабочей системы 9

2.2 Класс Main 10

2.3 Класс ConsoleSupporting 11

2.4 Класс Processor 12

**ВЫВОДЫ** 17

**СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ** 18

**ВВЕДЕНИЕ**

Основополагающей задачей практики было поставлено создание программного продукта (далее «ПП») поддержки компании-провайдера интернет-услуг.

Несмотря на бурное развитие Интернета и электронной коммерции (E-commerce) или хотя бы отдельных элементов последней, основной бизнес Интернет-провайдеров пока лежит в области предоставления услуг доступа к Сети. E-commerce занимает доли процента дохода, по сравнению с оборотами операторов связи или деятельности Интернет-провайдеров. На данный момент в России рынок интернет-провайдинга оценивается в 200-230 млн. долларов в год. Это сопоставимо с оборотами лишь одного крупного провайдера на Западе, а для России это большой рынок. Но, что особенно важно, вследствие малого вовлечения в Интернет, потенциал роста российских провайдеров многократно выше, нежели в Европе или США. Поэтому компании-провайдеры активно ищут доступ к пользователям. Разрабатываемый ПП будет особенно полезен тем, кто дает нам возможность работать в глобальной компьютерной Сети.

Краеугольным камнем для разработки является обширная база данных (далее «БД»), содержащая всю необходимую информацию: данные пользователей и клиентов, обращения клиентов, данные о финансовых операциях и др. После создания БД, разработано приложение для более комфортной работы с базой и поддержки других функций провайдера.

**ГЛАВА 1 РАБОТА НАД БАЗОЙ ДАННЫХ**

**1.1 Создание макета базы данных**

После получения информации об объектах, обработку и сохранение информации о которых необходимо будет содержать при работе с приложением, была разработана схема будущей БД (рис. 1.1). Основными таблицами являются:

* clients (id, name, technical\_information, id\_city, address, contact\_information);
* contracts (id, name, id\_client, preparation\_date, start\_date, id\_status, end\_date);
* users (id, last\_name, first\_name, login, password, organization, contact\_information, id\_role, id\_status);
* financial\_operations (id, id\_bill, id\_transaction\_type, sum, operation\_date, id\_user, id\_contract\_part, cancellation\_date, id\_cancellation\_user, comment).

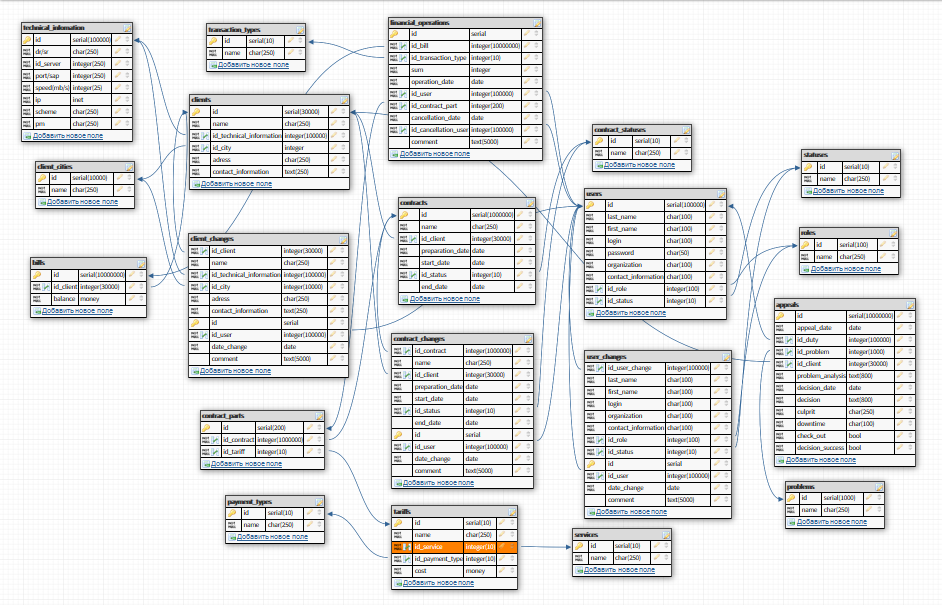


Рисунок 1.1 – Макет базы данных интернет-провайдера

**1.2 Реализация и наполнение информацией базы данных**

Для реализации созданной схемы БД была выбрана система управления базами данных (далее «СУБД») PostgreSQL, как более удобное и современное средство контроля баз данных. Для удобства работы был установлен интерфейс СУБД «pgAdmin», позволяющий работать с СУБД в графической среде.

Прежде всего был создан пользователь СУБД «A.T.A.N», через которого велась работа, с правами изменения лишь баз данных, для контроля доступа к информации. После чего развёрнута точка входа данного пользователя и создана новая БД «PracticeDB» (рис. 1.2).

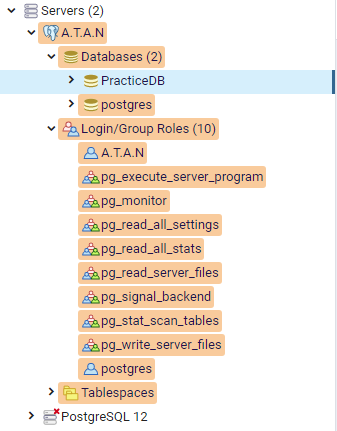


Рисунок 1.2 – Отображение начальных элементов

серверной системы СУБД

После создания базы, с сайта-конструктора была импортирована её структура, и, при помощи генератора тестовых данных, заполнены информацией созданные таблицы.

Также в ходе практики были созданы дополнительные не связанные с основной частью БД таблицы для практического изучения и тестирования создаваемых запросов и функций.

Результирующая БД содержит в себе 24 таблицы: 20 связанных основных и 4 дополнительных тестовых (рис. 1.3).

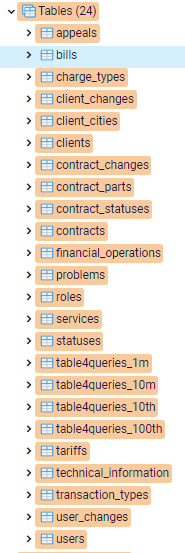


Рисунок 1.3 – Список таблиц базы данных «PracticeDB»

**1.3 Доработка функционала базы данных**

В базе данных был реализован триггер «Update\_bills()», отвечающий за обновление счетов клиентов при проведении финансовых операций, что улучшило автоматизацию системы и облегчило дальнейшую работу с БД. Код триггера показан на рисунке 1.4.

Также был создан и добавлен в планировщик заданий операционной системы backupping.bat-файл, отвечающий за резервное копирование базы данных для контроля изменений и ошибок в ней. Код из .bat-файла показан на рисунке 1.5.

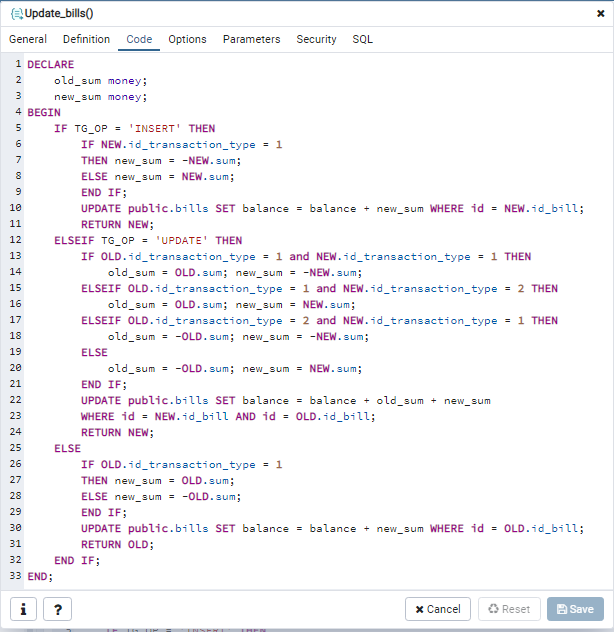


Рисунок 1.4 – Код триггера «Update\_bills()»

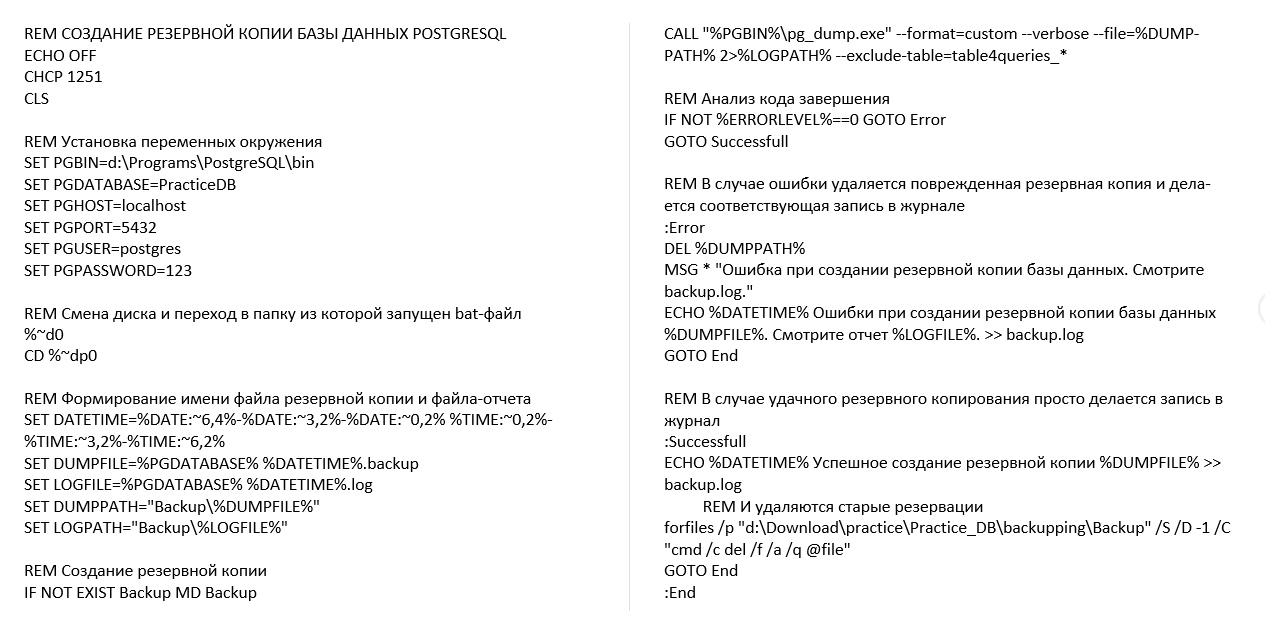


Рисунок 1.5 – Код backupping.bat-файла резервного копирования БД

**1.4 Запросы отчётов**

В отдельное задание было вынесено создание перекрёстных запросов, позволяющих выводить специальные таблицы отчётов, содержащих группированную статистику по конкретному критерию (рис. 1.6 и 1.7).

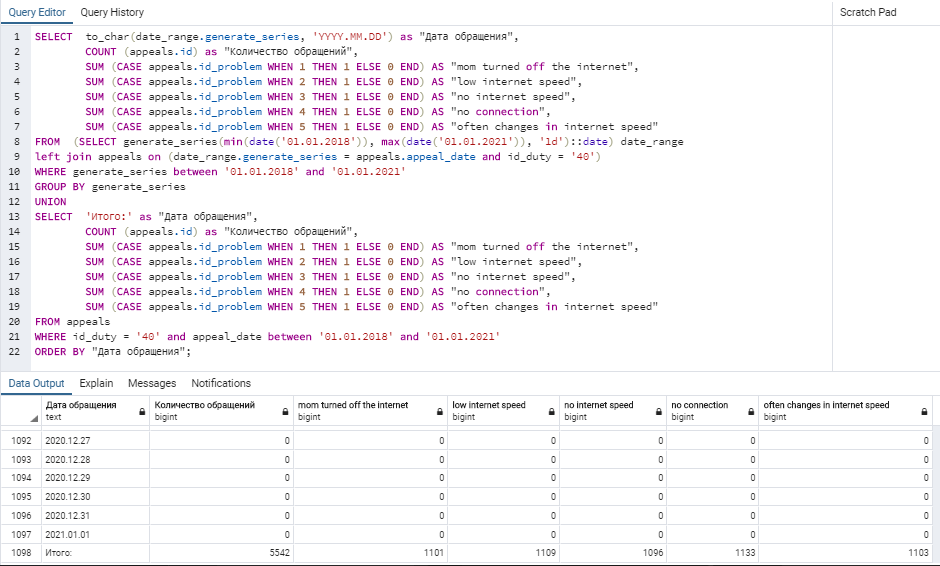


Рисунок 1.6 – Отчёт об обращениях

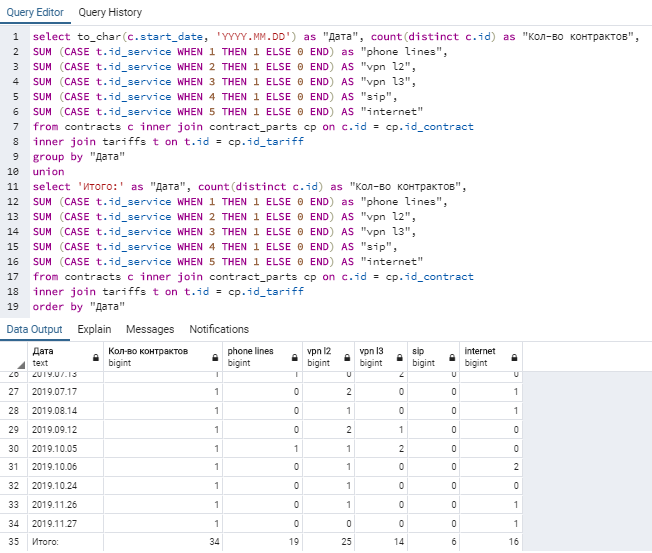


Рисунок 1.6 – Отчёт о контрактах

**ГЛАВА 2 РАБОТА НАД ПРИЛОЖЕНИЕМ**

**2.1 Преднастройка рабочей системы**

Для работы над будущим программным продуктом был выбран язык программирования Java, как один из самых современных и гибких языков высокого уровня (далее «ЯВУ»), и приложение IntelliJ IDEA, как интегрированная среда разработки программного обеспечения (далее «ПО»), созданная с расчётом на работу, прежде всего, с данным ЯВУ.

Для обеспечения контроля версий использован сервис GitHub (рис. 2.1). Был создан Maven-проект и загружен в новое хранилище на сервисе.

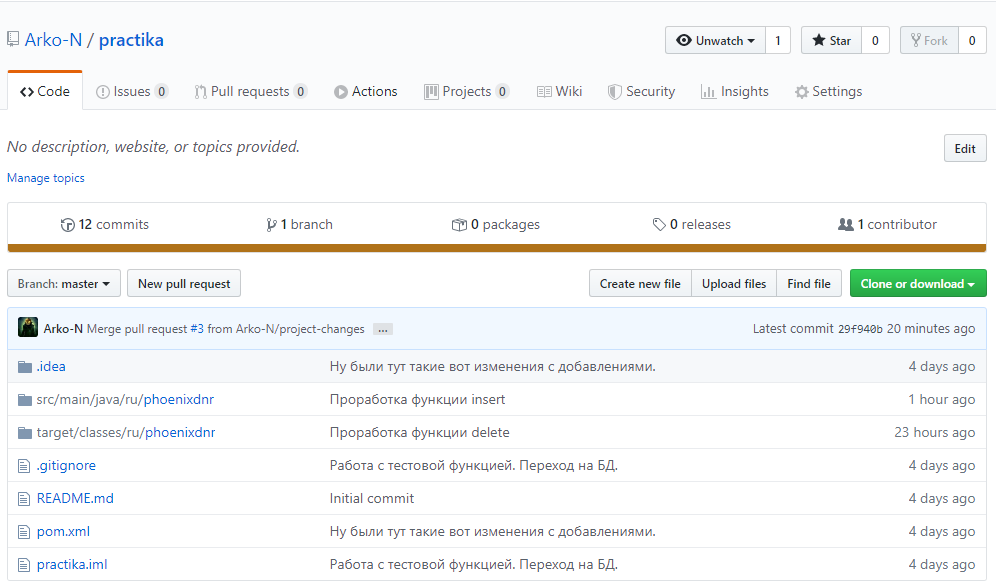


Рисунок 2.1 – Хранилище «practika» на GitHub’е

После всех приготовлений проект в IntelliJ IDEA был связан с системой контроля версий GitHub и базой данных PracticeDB (рис. 2.2).

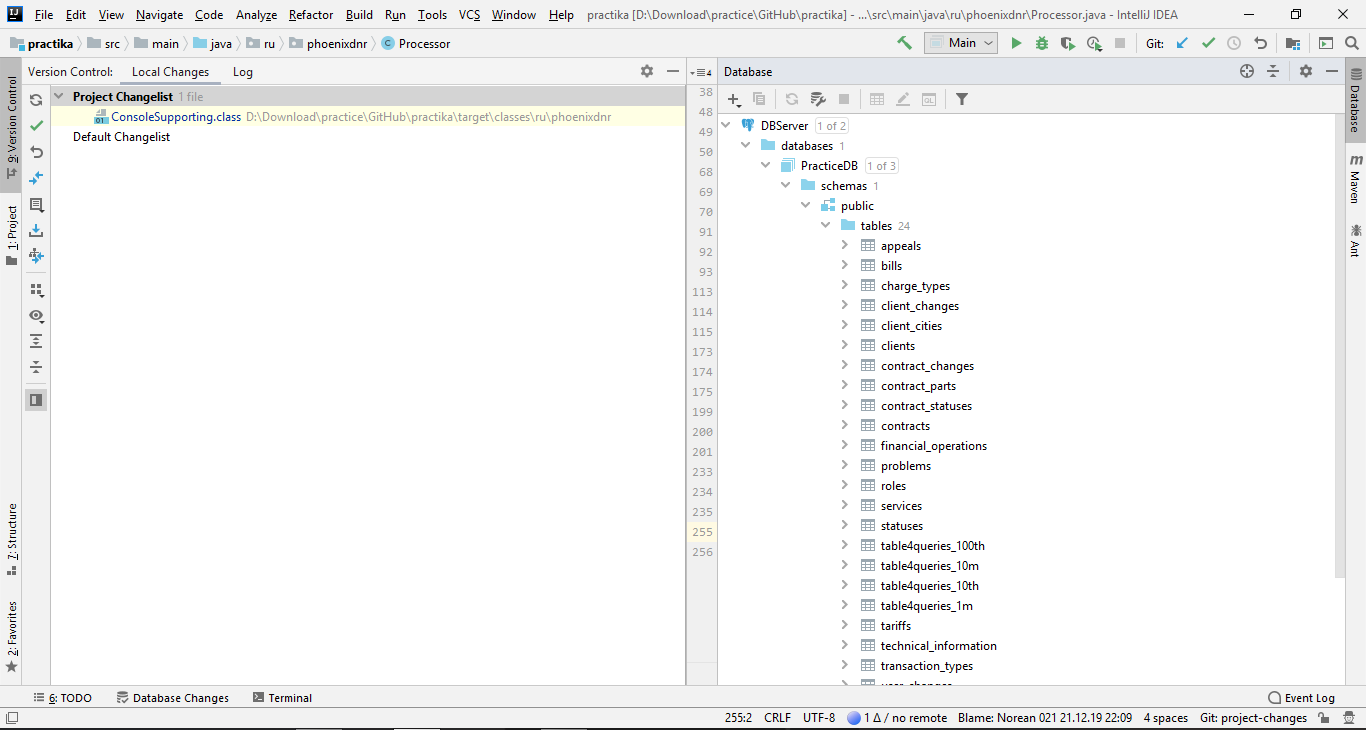


Рисунок 2.2 – Вкладки Version Control и Database в IntelliJ IDEA

Объединение всех систем в один проект завершил подготовку разработки ПП.

**2.2 Класс Main**

Класс Main – главный класс ПП, отвечающий за подготовку и запуск основной части приложения (рис. 2.3).

**package** ru.phoenixdnr;  
  
**import** java.io.IOException;  
  
**public class** Main  
{  
 **private** Main() {}  
  
 **public static void** main(String[] args) **throws** IOException  
 {  
 Processor processor = Processor.*getProcessor*();  
 processor.run();  
 }  
}

Рисунок 2.3 – Содержание класса Main

**2.3 Класс ConsoleSupporting**

Класс ConsoleSupporting содержит в себе ряд статических переменных и функций, необходимых для более корректной и быстрой работы с консолью как самого разработчика, так и конечного пользователя (рис. 2.4).

**package** ru.phoenixdnr;  
**import** java.util.Scanner;  
**public class** ConsoleSupporting {  
 **private** ConsoleSupporting() {}**protected static** String *input*;  
 **protected static** Scanner *scan* = **new** Scanner(System.***in***);

*//Strings* **protected static final** String ***QUIT*** = **"q"**;  
 **protected static final** String ***CANCEL*** = **"c"**;  
 **protected static final** String ***INSERT*** = **"INSERT"**;  
 **protected static final** String ***UPDATE*** = **"UPDATE"**;  
 **protected static final** String ***DELETE*** = **"DELETE"**;  
**protected static final** String ***VOID*** = **" "**;  
 **protected static final** String ***RUN*** = **"\nВведите \"RUN\" для начала работы\n"**;  
 **protected static final** String ***ABOUT\_TABLES*** = **"\nСписок таблиц:\n"**;  
 **protected static final** String ***SELECTION\_OF\_QUIT*** = **"\nВведите \"q\" чтобы закрыть приложение\n"**;  
 **protected static final** String ***SELECTION\_OF\_CANCELLING*** = **"\nВведите \"c\" для выхода на главную\n"**;  
 **protected static final** String ***SELECTION\_OF\_TABLE*** = **"\nВведите название таблицы, с которой будете работать\n"**;  
 **protected static final** String ***SELECTION\_OF\_PROCESSING*** = **"\nВведите INSERT для добавления пользователя\nВведите UPDATE для изменения пользователя\nВведите DELETE для удаления пользователя\n"**;  
 **protected static final** String ***INSERTING\_OF\_ROW*** = **"\nили данные для новой записи через запятую\n(строки и даты необходимо заключать в одиночные кавычки):\n"**;  
 **protected static final** String ***UPDATING\_ID*** = **"\nили id записи, которую нужно изменить:\n"**;  
 **protected static final** String ***UPDATING\_OF\_ROW*** = **"\nВведите новые данные для записи через запятую:\n"**;  
 **protected static final** String ***DELETING\_OF\_ROW*** = **"\nили id удаляемой записи:\n"**;  
 *//Queries* **protected static final** String ***SELECT\_TABLE\_NAME*** = **"SELECT table\_name FROM information\_schema.tables WHERE table\_schema='public' ORDER BY 1"**;  
 **protected static final** String ***SELECT\_COLUMN\_NAME*** = **"SELECT column\_name FROM information\_schema.columns WHERE table\_name='"**;  
 **protected static final** String ***SELECT\_ALL\_ROWS*** = **"SELECT \* FROM %s ORDER BY 1"**;  
 **protected static final** String ***INSERT\_ROW*** = **"INSERT INTO %s (%s) VALUES (%s)"**;  
 **protected static final** String ***UPDATE\_ROW*** = **"UPDATE %s SET %s WHERE id=%d"**;  
 **protected static final** String ***DELETE\_ROW*** = **"DELETE FROM %s WHERE id=%s;"**;  
 *//Functions* **protected static void** clearScreen() {  
 **for**(**int** i = 0; i < 5; i++) {  
 System.***out***.print(**"\n\n\n\n\n"**);  
 }  
 }  
}

Рисунок 2.4 – Содержание класса ConsoleSupporting

**2.4 Класс Processor**

Класс Processor – основной обрабатывающий класс приложения, содержащий функции для контроля работы ПП, взаимодействия с БД и вывода информации на экран (рис. 2.5 – 2.13).

**private static** Processor *processor*;  
**private** Processor() {}  
  
**public static synchronized** Processor getProcessor()  
{  
 **if** (*processor* == **null**) *processor* = **new** Processor();  
 **return** *processor*;  
}  
  
*//Variables***int counter** = 0;  
  
**private** Connection **connection**;  
**private** Statement **statement**;  
**private** ResultSet **resultSet**;  
  
**private** String **table**;  
**private** ArrayList **header** = **new** ArrayList();  
**private int columns**;

Рисунок 2.5 – Конструкторы и глобальные переменные класса Processor

*//Functions***public void** run()  
{  
 ConsoleSupporting.*clearScreen*();  
 System.***out***.println(ConsoleSupporting.***RUN***);  
 *input* = *scan*.nextLine();  
 **while** (!*input*.equals(ConsoleSupporting.***QUIT***))  
 {  
 setConnection();  
 processing();  
 }  
}

Рисунок 2.6 – Открытая функция, связывающая данный класс с остальными

**private void** setConnection()  
{  
 **try** {  
 Class.*forName*(**"org.postgresql.Driver"**);  
 **connection** = DriverManager.  
 *getConnection*(**"jdbc:postgresql://localhost:5432/PracticeDB"**, **"A.T.A.N"**, **"123"**);  
 **connection**.setAutoCommit(**false**);  
 System.***out***.println(**"-- Opened database successfully"**);  
  
 **statement** = **connection**.createStatement();  
 }  
 **catch** (Exception e)  
 {  
 e.printStackTrace();  
 System.***err***.println(e.getClass().getName() + **": "** + e.getMessage());  
 System.*exit*(0);  
 }  
}

Рисунок 2.7 – Функция обеспечения связи с базой данных

**private void** printTableHead()  
{  
 **try** {  
 **columns** = 0;  
 **header**.clear();  
 **resultSet** = **statement**.executeQuery(ConsoleSupporting.***SELECT\_COLUMN\_NAME*** + **table** + **"'"**);  
 **while** (**resultSet**.next())  
 {  
 **columns**++;  
 **header**.add(**resultSet**.getString(1).trim());  
 System.***out***.printf(**"%.15s\t"**, **resultSet**.getString(1).trim() + ConsoleSupporting.***VOID***);  
 }  
 System.***out***.println();  
 }  
 **catch** (Exception e)  
 {  
 e.printStackTrace();  
 System.***err***.println(e.getClass().getName() + **": "** + e.getMessage());  
 System.*exit*(0);  
 }  
}

Рисунок 2.8 – Функция поддержки вывода «шапки» таблиц

**private void** printTableRows()  
{  
 **try** {  
 **resultSet** = **statement**.executeQuery(String.*format*(ConsoleSupporting.***SELECT\_ALL\_ROWS***, **table**));  
 **while** (**resultSet**.next())  
 {  
 **for** (**int** i = 1; i <= **columns**; i++)  
 {  
 System.***out***.printf(**"%.15s\t"**, String.*valueOf*(**resultSet**.getObject(i)).trim() + ConsoleSupporting.***VOID***);  
 }  
 System.***out***.println();  
 }  
 }  
 **catch** (Exception e)  
 {  
 e.printStackTrace();  
 System.***err***.println(e.getClass().getName() + **": "** + e.getMessage());  
 System.*exit*(0);  
 }  
}

Рисунок 2.9 – Функция поддержки вывода строк таблиц

**private void** processing() {  
 **try** {  
 ConsoleSupporting.*clearScreen*();  
 System.***out***.println(ConsoleSupporting.***ABOUT\_TABLES***);  
 **resultSet** = **statement**.executeQuery(ConsoleSupporting.***SELECT\_TABLE\_NAME***);  
 **counter** = 0;  
 **while** (**resultSet**.next()) {  
 **if** (**counter** == 4) {System.***out***.println(); **counter** = 0;}  
 System.***out***.printf(**"%.30s\t"**, **resultSet**.getString(1).trim() + ConsoleSupporting.***VOID***);  
 **counter**++;  
 }  
 System.***out***.println();  
 System.***out***.print(ConsoleSupporting.***SELECTION\_OF\_QUIT***);  
 System.***out***.println(ConsoleSupporting.***SELECTION\_OF\_TABLE***);  
 *input* = *scan*.nextLine();  
 **if** (!*input*.equals(ConsoleSupporting.***QUIT***)) {  
 **table** = *input*;  
 ConsoleSupporting.*clearScreen*();  
 printTableHead();  
 printTableRows();  
 System.***out***.print(ConsoleSupporting.***SELECTION\_OF\_CANCELLING***);  
 System.***out***.println(ConsoleSupporting.***SELECTION\_OF\_PROCESSING***);  
 *input* = *scan*.nextLine();  
 **if** (*input*.equals(ConsoleSupporting.***INSERT***)) {  
 **while** (!*input*.equals(ConsoleSupporting.***CANCEL***)) {  
 insert();  
 }  
 }  
 **if** (*input*.equals(ConsoleSupporting.***UPDATE***)) {  
 **while** (!*input*.equals(ConsoleSupporting.***CANCEL***)) {  
 update();  
 }  
 }  
 **if** (*input*.equals(ConsoleSupporting.***DELETE***)) {  
 **while** (!*input*.equals(ConsoleSupporting.***CANCEL***)) {  
 delete();  
 }  
 }  
 }  
 **resultSet**.close();  
 **statement**.close();  
 **connection**.commit();  
 **connection**.close();  
 }  
 **catch** (Exception e) {  
 e.printStackTrace();  
 System.***err***.println(e.getClass().getName() + **": "** + e.getMessage());  
 System.*exit*(0);  
 }  
}

Рисунок 2.10 – Функция, представляющая собой основу работы класса

**private void** insert()  
{  
 **try** {  
 ConsoleSupporting.*clearScreen*();  
 System.***out***.print(ConsoleSupporting.***SELECTION\_OF\_CANCELLING***);  
 System.***out***.println(ConsoleSupporting.***INSERTING\_OF\_ROW***);  
  
 String string = String.*valueOf*(**header**);  
 System.***out***.println(string.substring(5, string.length() - 1));  
 System.***out***.println();  
  
 *input* = *scan*.nextLine();  
 **if** (!*input*.equals(ConsoleSupporting.***CANCEL***))  
 {  
 **statement**.executeUpdate(String.*format*(ConsoleSupporting.***INSERT\_ROW***, **table**, string.substring(5, string.length() -1), *input*));  
 }  
 }  
 **catch** (Exception e)  
 {  
 e.printStackTrace();  
 System.***err***.println(e.getClass().getName() + **": "** + e.getMessage());  
 System.*exit*(0);  
 }  
}

Рисунок 2.11 – Функция, обеспечивающая добавление записей в таблицы БД

**private void** update() {  
 **try** {  
 ConsoleSupporting.*clearScreen*();  
 System.***out***.print(ConsoleSupporting.***SELECTION\_OF\_CANCELLING***);  
 System.***out***.println(ConsoleSupporting.***UPDATING\_ID***);  
 *input* = *scan*.nextLine();  
 **if** (!*input*.equals(ConsoleSupporting.***CANCEL***)) {  
 **int** id = Integer.*parseInt*(*input*);  
 System.***out***.println(ConsoleSupporting.***UPDATING\_OF\_ROW***);  
 printTableHead();  
 System.***out***.println();  
 *input* = *scan*.nextLine();  
 String[] inputted = *input*.split(**", "**);  
 AtomicReference<String> query = **new** AtomicReference<>();  
 **counter** = 0;  
 **header**.forEach((v) -> {query.set(query.get() + v + **"='"** + inputted[**counter**] + **"', "**); **counter**++;});  
 String string = query.toString();  
 **statement**.executeUpdate(String.*format*(ConsoleSupporting.***UPDATE\_ROW***, **table**, string.substring(0, string.length() - 2), id));  
 } }  
 **catch** (Exception e) {  
 e.printStackTrace();  
 System.***err***.println(e.getClass().getName() + **": "** + e.getMessage());  
 System.*exit*(0);  
 }  
}

Рисунок 2.12 – Функция, обеспечивающая изменение записей в таблицах БД

**private void** delete()  
{  
 **try** {  
 ConsoleSupporting.*clearScreen*();  
 System.***out***.print(ConsoleSupporting.***SELECTION\_OF\_CANCELLING***);  
 System.***out***.println(ConsoleSupporting.***DELETING\_OF\_ROW***);  
  
 *input* = *scan*.nextLine();  
 **if** (!*input*.equals(ConsoleSupporting.***CANCEL***))  
 {  
 **statement**.executeUpdate(String.*format*(ConsoleSupporting.***DELETE\_ROW***, **table**, *input*));  
 }  
 }  
 **catch** (Exception e)  
 {  
 e.printStackTrace();  
 System.***err***.println(e.getClass().getName() + **": "** + e.getMessage());  
 System.*exit*(0);  
 }  
}

Рисунок 2.13 – Функция, обеспечивающая удаление записей из таблиц БД

Взаимодействие приложения с пользователем и вывод информации реализованы в консольном виде (рис. 2.14).

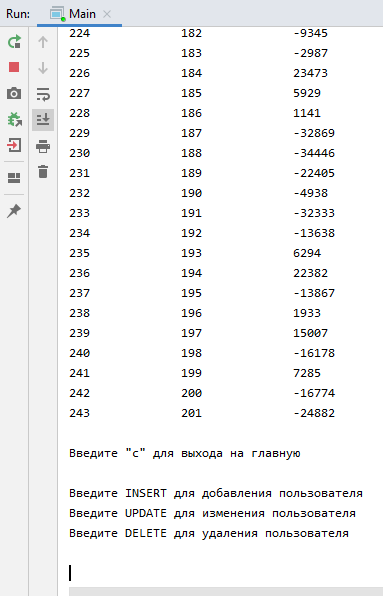


Рисунок 2.14 – Пример взаимодействия с запущенным приложением

**ВЫВОДЫ**

В заключение отметим, что цели производственной практики достигнуты. Мы провели исследования, осуществили полный цикл разработки приложения, от первоначального исследования предметной области до создания самого программного продукта. Для этого мы использовали pgAdmin (PostgreSQL), IntelliJ IDEA (Java) и GitHub (контроль версий).

Провели максимально приближенный к реальному цикл разработки приложения. Использовали полученные ранее знания, а также результаты исследований.

Построили базу данных на основе современными средств управления, таких как PostgreSQL, для того, чтобы определить ключевые функции в работе интернет-провайдера.

Само приложение было создано с использованием одного из самых известных и популярных языков программирования высокого уровня в мире – Java.

Убежден, что полученные знания и опыт мы сможем применить в дальнейшем трудоустройстве.

**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

1. Грекул В.И. - Проектирование информационных систем- ИНТУИТ.ру, 2009
2. Э.К. Алгазинов, А.А. Сирота. Анализ и компьютерное моделирование информационных процессов и систем. - М.: Диалог-МИФИ, 2009. - 416 с.
3. И.В. Соловьев, А.А. Майоров. Проектирование информационных систем. - М.: Академический Проект, 2009. - 400 с.
4. В.Ю. Пирогов. Информационные системы и базы данных. Организация и проектирование. - СПб.: БХВ-Петербург, 2009. - 528 с.
5. К.Н. Мезенцев. Автоматизированные информационные системы. - М.: Академия, 2012. - 174 с.
6. Н.З. Емельянова, Т.Л. Партыка, И.И. Попов. Устройство и функционирование информационных систем. - М.: Форум, Инфра-М, 2012. - 448 с.
7. А.Н. Калашян, Г.Н. Калянов. Структурные модели бизнеса: DFD-технологии. - М.: Прикладные информационные технологии, 2009. - 256 с.