

МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ФАКУЛЬТЕТ ПРИКЛАДНОЙ МАТЕМАТИКИ И ИНФОРМАТИКИ
Кафедра многопроцессорных систем и сетей

Спецификация банковского приложения
(групповой отчет)

Преподаватель
Довнар С.Е.

Минск, 2014

Оглавление

[1.Команда](#)

[2.Общие требования](#)

[2.Модель базы данных](#)

[3.Описание User-stories](#)

[4.Work by People](#)

[Сачок Илья.](#)

[Понтелей Виталий.](#)

[Гилевич Павел.](#)

[Будько Алексей.](#)

[Захарова Анна.](#)

[Рымарчик Александр.](#)

[Литвинов Владимир.](#)

[Алтыев Мурад.](#)

Ссылка на гитхаб: https://github.com/vitalbit/bank_app

1. Команда

- 1) Алтыев Мурад
- 2) Будько Алексей
- 3) Гилевич Павел
- 4) Захарова Анна
- 5) Литвинов Владимир
- 6) Понтелей Виталий
- 7) Рымарчик Александр
- 8) Сачок Илья
- 9) Хакназаров Зульфикор

2. Общие требования

Цель проекта -- банковское консольное приложение для операций в банке. Продукт является независимым приложением и создается в ознакомительных целях. У приложения должна существовать база данных клиентов банка. Все клиенты банка могут вкладывать деньги и снимать их со своего счета, также проверять баланс, оплачивать телефонные услуги и осуществлять перевод денежных средств на другие счета. Аудитория приложения -- администратор и операционист. Приложение должно взаимодействовать с базой данных и предлагать запрограммированные операции связанные с выборкой данных, обновлением, созданием и удалением (напр. вывод всех пользователей на экран). Также должна существовать аутентификация при запуске приложения.

2. Модель базы данных

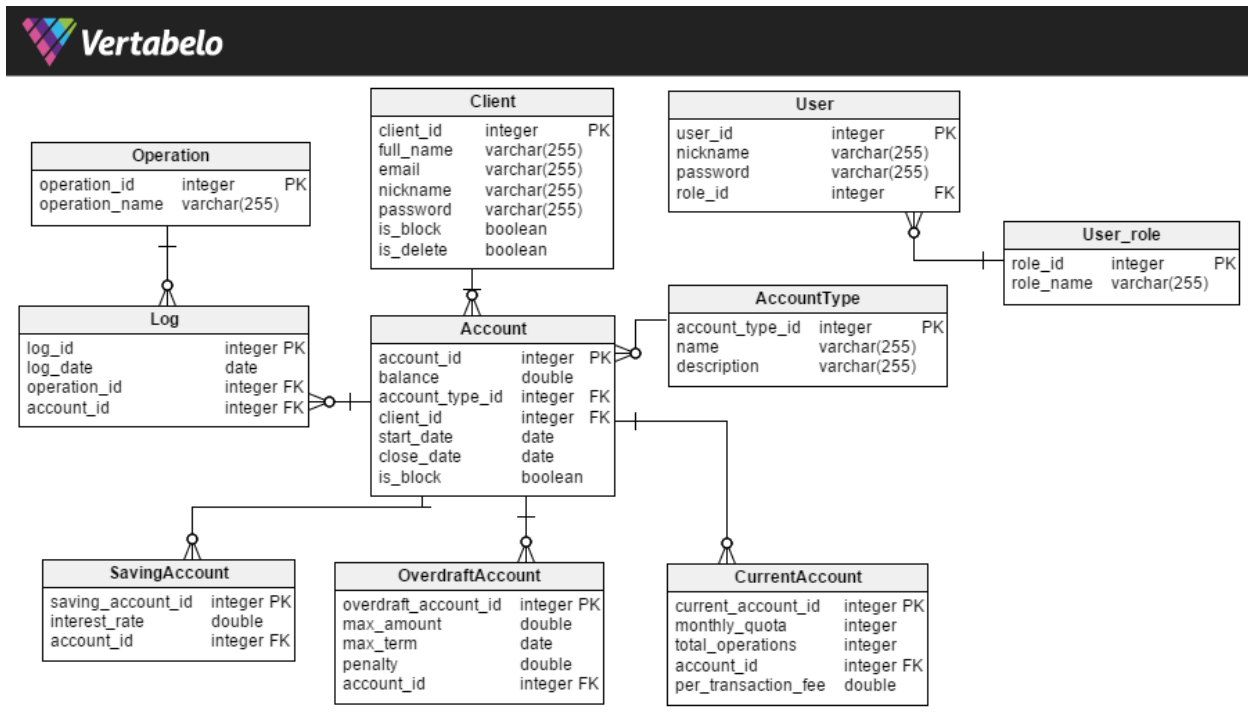


Таблица Client содержит информацию о клиентах. Таблица Account содержит информацию о счетах клиентов и является прародителем для таблиц, описывающих отдельный тип счета. Также в БД хранится история операций в таблице Log. Таблица User предназначена для аутентификации, она будет содержать всех администраторов и операторов.

3. Описание User-stories

Приложение должно предлагать следующие возможности:

- Добавление, удаление пользователя из банковской системы
- Просмотр и корректировка пользовательских данных
- Создание и удаление счета для определенного клиента
- Блокировка и разблокировка счетов клиента
- Просмотр и корректировка данных о счете
- Просмотр истории транзакций
- Возможность вложить определенную сумму на счет
- Возможность снять определенную сумму со счета

Как администратор я могу добавить нового пользователя в систему.

Все что нам необходимо, это выбрать соответствующий пункт в меню, и ввести запрашиваемые данные, а именно: логин, пароль, полное имя и электронную почту. В случае если операция по созданию пользователя прошла успешно, то вы получите

соответствующее сообщение, иначе, сообщение с описанием ошибки (например если в системе уже существует пользователь с таким логином). Операция может выполняться администратором, если нужно создать нового пользователя.

Как администратор я могу удалить пользователя из системы.

Для этого надо знать id пользователя. Далее администратор ставит флаг над полем is_delete, показывая, что пользователь удален, но остается храниться в архиве.

Как администратор я могу корректировать данные о пользователе.

Для данной user-story необходимо знать id пользователя. Далее администратору предлагается выбрать один из трех параметров для редактирования (полное имя пользователя, его никнейм или электронная почта), а затем ввести новое значение параметра.

Как администратор я могу посмотреть информацию о пользователе.

Для выполнения данной user-story также понадобится только id пользователя. Будет отображаться следующая информация: полное имя клиента, почта, заблокирован ли его счет, а также ник и пароль.

Как администратор я могу восстановить клиента.

По id пользователя администратор снимает флаг с поля is_delete и возвращает право клиенту пользоваться его счетами.

Как администратор я могу заблокировать(разблокировать) счет.

-- Используя id счета необходимо обеспечить блокировку счета пользователя в случае потери им карточки.

Как администратор я могу просматривать состояние счета пользователя.

--- Зная id счета оператор должен получить возможность просмотреть всю доступную информацию о соответствующем счете.

Как администратор я могу создавать новый счет пользователя.

Зная id пользователя, администратор может создать для него один из трех видов счета: сберегательный, текущий или счет с использованием овердрафта. Далее администратор задает специфические параметры для выбранного вида счета (процентная ставка и т.п.).

Как оператор я могу просматривать историю транзакций счета.

-- При помощи id счета оператор может узнать все операции, которые выполнялись по отношению к счету.

--

Как оператор я могу вкладывать деньги, которые принес мне клиент.

-- Клиент приходит в банк и дает некоторую сумму денег оператору, а также говорит ему счет, на который он хочет положить деньги. Номер счета соответствует идентификатору в таблице Accounts, таким образом оператор вводит номер счета, денежную сумму и в итоге ему выдается ответ прошла ли операция.

Как оператор я могу снять со счета клиента сумму, которую он пришел снять. Когда клиент пришел в банк, он проходит к одному из свободных операторов, называет оператору id счета и некоторую сумму денег, которую нужно снять со счета. Операционист проверяет есть ли указанная сумма на счете и выдает её, если она есть. В противном случае просит её подкорректировать в меньшую сторону. Попутно операционист делает соответствующую запись в журнале о данной операции.

Как оператор я могу просмотреть баланс.

Клиент хочет узнать сколько денег на его счету, например перед каким-либо платежом. Для этого оператор выбирает соответствующий пункт в меню и вводит номер счета пользователя. Результат - сумма на счету.

4.Work by People

Сачок Илья.

Была написана начальная версия main.c.

Пример команды ввыводы информации в консоль:

// Main print

```
static int printResult(void *data, int argc, char **argv, char **azColName) {
    int i;
    fprintf(stderr, "%s: ", (const char*)data);
    for (i = 0; i<argc; i++){
        printf("%s = %s\n", azColName[i], argv[i] ? argv[i] : "NULL");
    }
    printf("\n");
    return 0;
}
```

Были сделаны гибкие правила для добавления дополнительного функционала. В зависимости от количество записей в массиве:

// Put your operations here

```
char *states[2] = {
    "1. See all account.",
    "2. Credit money."
};
```

на консоль печатаются все доступные операции и в switch statement выполняется указанная команда.

```
switch (operation) {
```

```

case 1:
    rc = getAccountInfoById(db, zErrMsg);
    break;
case 2:
    if (credit(db))
        printf("Credit success!\n");
    else
        printf("Credit error!\n");
    break;
}

```

Был написан makefile для автоматизации сборки приложения и его запуска.

```

# c compiler
# CC=$(CROSS_COMPILE)clang
CC=$(CROSS_COMPILE)gcc

# Variables
DB=Bank.sqlite
LIB=lib
SRC=$(wildcard include/*.c)

CMD_ALL=$(CC) $^ -o bank

ifeq ($(OS), Windows_NT)
    CMD_RUN=bank.exe $(DB)
else
    UNAME_S := $(shell uname -s)
    ifeq ($(UNAME_S),Linux)
        CMD_ALL += -lpthread -ldl
    endif
    CMD_RUN=./bank $(DB)
endif

# default make task
all: main.c $(SRC) $(LIB)/sqlite3.c
    $(CMD_ALL)

# start application
run: bank
    $(CMD_RUN)

# delete executable files
clean: bank
    rm bank

```

Для Continuous integration был настроен <https://travis-ci.org>

The image shows two screenshots of the Travis CI web interface. The top screenshot displays the user profile for 'Ilia Sachok', showing a list of repositories with toggle switches to enable or disable them. The bottom screenshot shows the build status for the repository 'vitalbit/bank_app', indicating a successful build with 9 tests passed. A build matrix table is also visible, showing the build configuration for different jobs.

Top Screenshot: User Profile

Travis CI for Private Repositories

Ilia Sachok

Accounts

Ilia Sachok
Repositories: 16

Repositories

We're only showing your public repositories below. You can find your private projects on travis-ci.com.
Enable your projects below by flicking the switch, add a `.travis.yml` to your project, and push a new commit to GitHub.

Last synchronized from GitHub: 18 minutes ago [Sync now](#)

Repository	Status
IFours/Accounts	OFF
IFours/amazon-ses	OFF
IFours/bank_app	ON
IFours/bank_app	ON
IFours/Calendar-PhoneGap-Plugin	OFF
IFours/cinema-card-online	OFF
IFours/DSS	OFF

Bottom Screenshot: Repository Build Status

Travis CI for Private Repositories

Ilia Sachok

Search all repositories

My Repositories Recent +

- vitalbit/bank_app 9
27 sec
about a minute ago
- IFours/bank_app
- IFours/bank_app 21
1 min 30 sec
about 5 hours ago

vitalbit/bank_app build passing Settings

Current Build History Pull Requests Branch Summary

master - fixes

#9 passed

- ran for 27 sec
- about a minute ago
- Commit 0c4996e
- Compare 464ef84...0c4996e

Vitalii authored and committed

Build Matrix

Job	Duration	Finished	Compiler	OS
9.1	14 sec	about a minute ago	gcc	linux
9.2	13 sec	about a minute ago	clang	linux

The screenshot displays the Travis CI web interface for the repository `vitalbit/bank_app`. The top navigation bar includes links for Home, Blog, Status, and Help, along with the Travis CI logo and the user profile of Ilia Sachok. The main content area is divided into two sections: a left sidebar for repository management and a right section for build details.

The left sidebar shows a search bar and a list of repositories. The repository `vitalbit/bank_app` is highlighted, showing 9 builds. Below it, the repository `IFours/bank_app` is listed with 21 builds. The right section shows the build history for `vitalbit/bank_app`, with a status of `build passing`. The build history table lists the following builds:

Build	Message	Commit	Duration	Finished
9	fixes	0c4996e (master)	27 sec	2 minutes ago
8	Delete createClientAndPrintBalance.h	464ef84 (master)	39 sec	18 minutes ago
7	Update header.h	489eeda (master)	33 sec	22 minutes ago
6	Rename delete&restoreClientByClientID.c to deleteAndRestoreClientByClientID.c	977fdeb6 (master)	31 sec	24 minutes ago
5	refactoring	8ccbd88 (master)	17 sec	28 minutes ago
4	Update .travis.yml	7005f0d (master)	20 sec	32 minutes ago
3	Update README.md	41ac969 (master)	23 sec	34 minutes ago
2	Delete #makefile#	4985573 (master)	24 sec	35 minutes ago
1	Create .travis.yml	c2c5c66 (master)	19 sec	36 minutes ago

В качестве фреймворка для тестирования был выбран: <http://check.sourceforge.net/>

Пример:

```
START_TEST(test_getAccountInfoById_create)
```

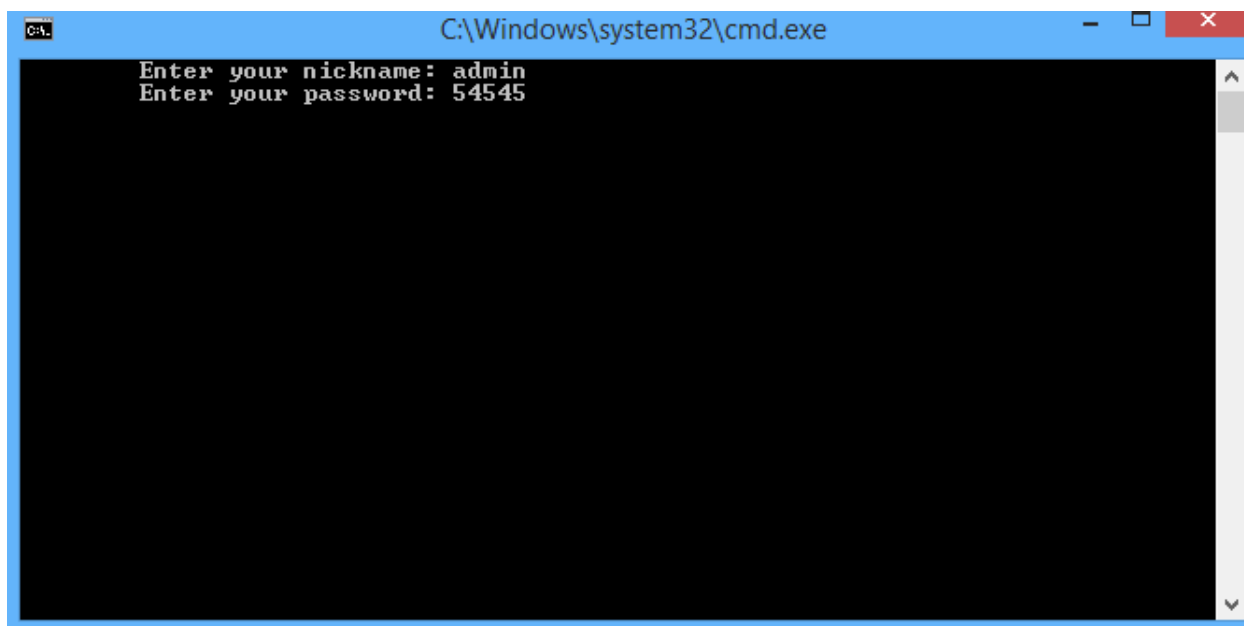
```
{
    char *zErrMsg = 0;
    char id[100] = "2";
    int rc = 0;
    sqlite3 *db;
    sqlite3_open("../Bank.sqlite", &db);
```

```
    rc = getAccountInfoById(db, zErrMsg, id);
    ck_assert_int_eq(rc, 0);
}
```

```
END_TEST
```

Понтелей Виталий.

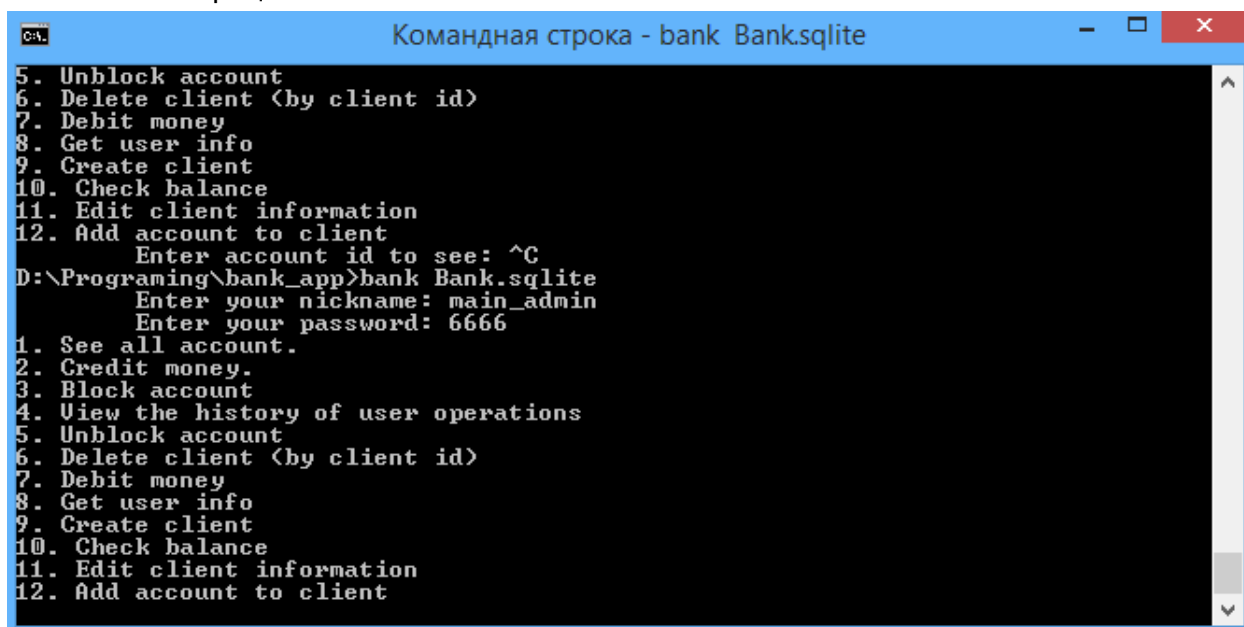
Был создан метод авторизации пользователя при входе в систему. При запуске программы предлагается ввести имя пользователя и пароль.



```
C:\Windows\system32\cmd.exe
Enter your nickname: admin
Enter your password: 54545
```

Авторизация

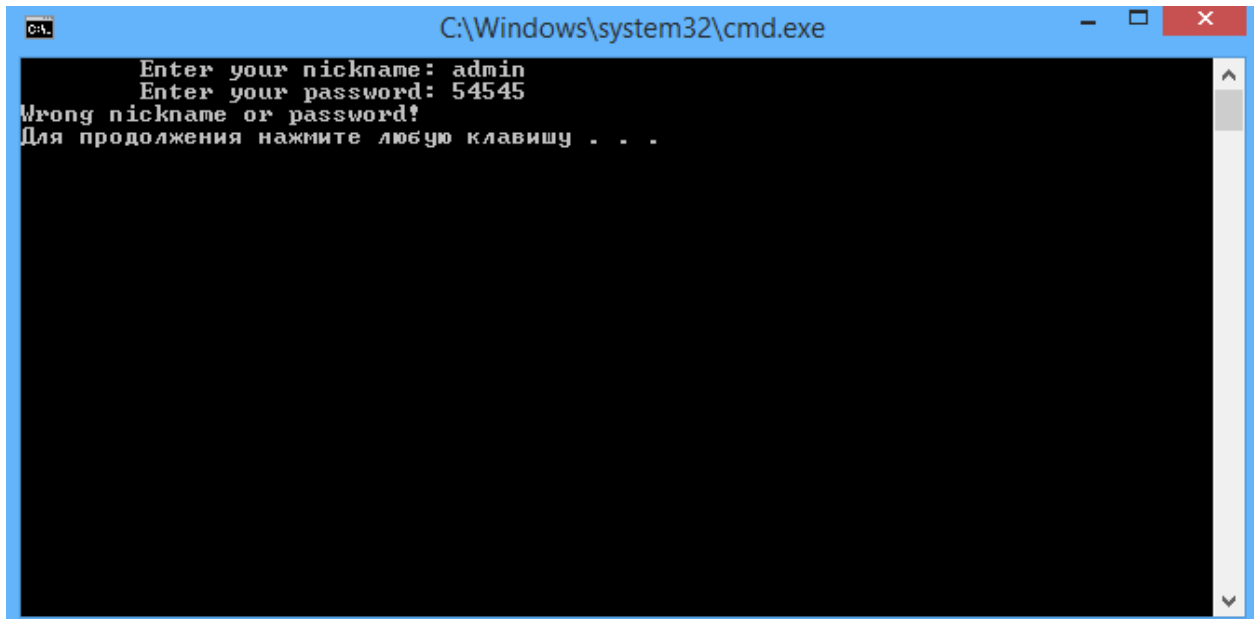
Если введенные данные соответствуют данным в базе данных, то пользователю открывается окно операций.



```
Командная строка - bank Bank.sqlite
5. Unblock account
6. Delete client <by client id>
7. Debit money
8. Get user info
9. Create client
10. Check balance
11. Edit client information
12. Add account to client
    Enter account id to see: ^C
D:\Programing\bank_app>bank Bank.sqlite
Enter your nickname: main_admin
Enter your password: 6666
1. See all account.
2. Credit money.
3. Block account
4. View the history of user operations
5. Unblock account
6. Delete client <by client id>
7. Debit money
8. Get user info
9. Create client
10. Check balance
11. Edit client information
12. Add account to client
```

Меню программы

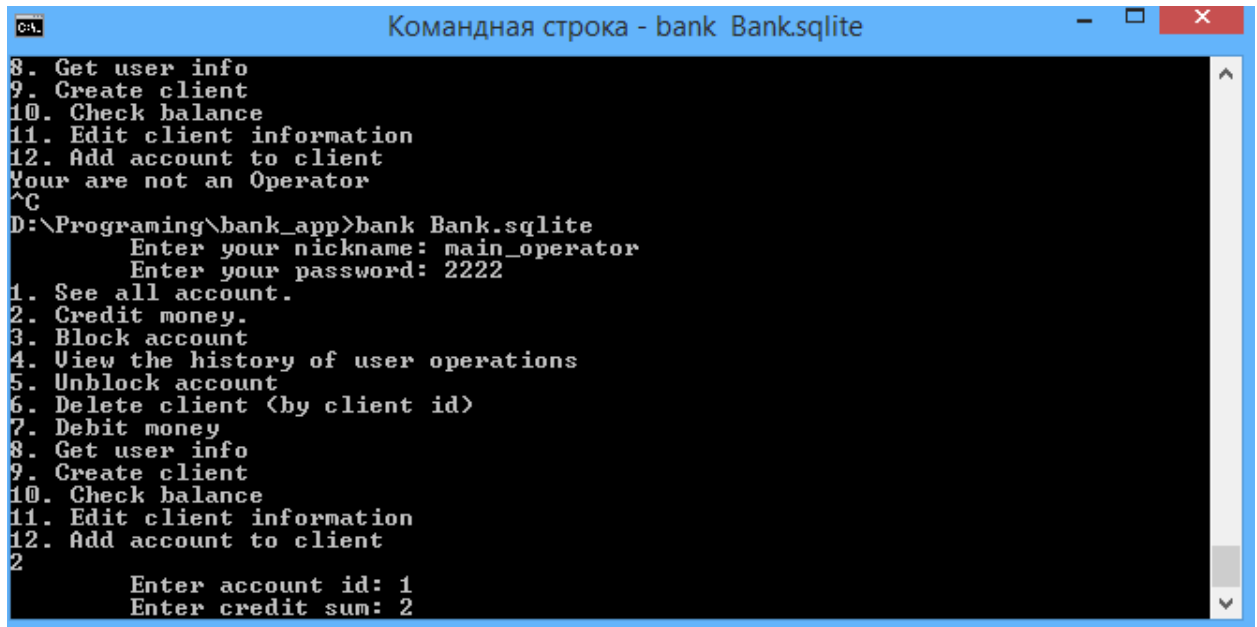
При неправильно введенных данных система сообщит, что данные введены неверно.



Неверно введенные данные

Когда пользователь авторизовался, из базы данных выбирается информация о роли авторизованного пользователя (либо администратор, либо операционист). Она сохраняется в глобальную переменную, которая потом используется для определения доступных пунктов меню для определенного пользователя.

Был создан метод `credit` позволяющий вносить деньги на счет. Деньги на счет может вносить только оператор. Поэтому вначале стоит проверка на роль пользователя, а также проверка не является ли счет заблокированным, для чего была создана функция `isAccountBlock`. Причем, если счет заблокирован, то система сообщит об этом и не позволит проводить операцию со счетом. Затем предлагается ввести счет клиента и сумму, которую он хочет вложить.



```
8. Get user info
9. Create client
10. Check balance
11. Edit client information
12. Add account to client
Your are not an Operator
^C
D:\Programing\bank_app>bank Bank.sqlite
    Enter your nickname: main_operator
    Enter your password: 2222
1. See all account.
2. Credit money.
3. Block account
4. View the history of user operations
5. Unblock account
6. Delete client (by client id)
7. Debit money
8. Get user info
9. Create client
10. Check balance
11. Edit client information
12. Add account to client
2
    Enter account id: 1
    Enter credit sum: 2
```

Вклад денежной суммы

При успешной операции система напишет, что операция прошла успешно, в противном случае напишет в чем проблема.

Поскольку существует несколько типов счетов, необходимо было определить, что это за счет, и в зависимости от типа, производить операции с ним. Так например Текущий счет в отличии от Сберегающего, еще обновляет количество произведенных операций, а счет предоставляющий овердрафт, проверяет не превышает ли новая сумма допустимый максимум.

Для ведения хронологии по вкладам создан метод creditLog, который добавляет запись в таблицу Log.

Гилевич Павел.

As an administrator, I can edit user

Для реализации первой user-story была создана функция

```
void editClient(sqlite3 *db, int client_id, int field_num)
```

В качестве параметров передаются дескриптор базы данных, уникальный идентификатор клиента, а также номер поля, подлежащего редактированию.

Приведем пример части кода, изменяющего полное имя пользователя:

```

switch (field_num)
{
case 1:
    printf("Input new full name: ");
    fflush(stdin);
    fgets(full_name, 79, stdin);
    query = "update Client set full_name = ? where client_id = ?";
    sqlite3_prepare_v2(db, query, strlen(query), &stmt, NULL);
    sqlite3_bind_text(stmt, 1, full_name, strlen(full_name), SQLITE_STATIC);
    sqlite3_bind_int(stmt, 2, client_id);
    if (sqlite3_step(stmt) == SQLITE_DONE)
    {
        printf("Full Name updated!! New Full Name is %s\n", full_name);
    }
    sqlite3_finalize(stmt);
    break;
}

```

Для изменения других полей производятся те же действия, за исключением небольшого изменения строки запроса query.

Приведем скриншот, демонстрирующий работу функции:

```

1. See all account.
2. Credit money.
3. Block an account (by client id)
4. Block an account (by nickname)
5. View the history of user operations
6. Edit client information
7. Add account to client
6
Input client's ID: 2
Select field to edit:
    1 - Full name
    2 - Nickname
    3 - Email
    0 - Exit
3
Input new email: mymail@gmail.com
Full Name updated!! New Email is mymail@gmail.com

```

As an administrator, I can add account to user

Для реализации второй user-story использовалась функция

```

void addAccountToClient(sqlite3 *db, int client_id, int
                        acc_type_id)

```

В качестве параметров она принимает дескриптор базы данных, идентификатор клиента и идентификатор, определяющий тип создаваемого счета.

На первом этапе происходит создание счета в главной таблице Accounts, так как для добавления счета в отдельные таблицы для каждого типа, необходимо указать его идентификатор в главной.

Код, выполняющий данное действие, приведен ниже:

```
printf("Input start balance: ");
scanf("%lf", &balance);
printf("Input open date (yyyy-mm-dd): ");
scanf("%s", date);

query = "insert into Account (balance, account_type_id, client_id, start_date, close_date, is_block) values (?, ?, ?, ?, ?, ?)";
sqlite3_prepare_v2(db, query, strlen(query), &stmt, NULL);
sqlite3_bind_double(stmt, 1, balance);
sqlite3_bind_int(stmt, 2, acc_type_id);
sqlite3_bind_int(stmt, 3, client_id);
sqlite3_bind_text(stmt, 4, date, strlen(date), SQLITE_STATIC);
sqlite3_bind_text(stmt, 5, "", 0, SQLITE_STATIC);
sqlite3_bind_int(stmt, 6, 0);
sqlite3_step(stmt);
sqlite3_finalize(stmt);
```

Далее происходит задание параметров, специфичных для каждого типа счета, а затем добавление его в таблицу, соответствующую типу. При этом, поскольку данные счетов из главной таблицы не удаляются даже в случае, если счет закрывается, то в качестве идентификатора выше добавленного счета используется количество записей в таблице Accounts (так как нужный нам счет был добавлен в нее последним).

Пример кода создания Сберегательного счета (Savings Account):

```
switch (acc_type_id)
{
case 1:
    printf("Input interest rate: ");
    scanf("%lf",&interest_rate);

    account_query = "insert into SavingAccount (interest_rate, account_id) values (?,(SELECT COUNT (*) FROM Account))";
    sqlite3_prepare_v2(db, account_query, strlen(account_query), &stmt, NULL);
    sqlite3_bind_double(stmt, 1, interest_rate);
    if (sqlite3_step(stmt) == SQLITE_DONE)
        printf("Savings Account added succesfully.\n");
    else
        printf(sqlite3_errmsg(db));
    sqlite3_finalize(stmt);
    break;
}
```

Будько Алексей.

После распределения заданий, необходимо было реализовать следующие User-story:

- As an administrator i can get information about user.
- As an operator i can debit user's money.

Распределение можно посмотреть по следующей ссылке:

<https://insurg.tpondemand.com/RestUI/Board.aspx?invite&acid=6C17D8319C81AC3D36AFA D64CAE08A28#page=board/5347746876595649827&appConfig=eyJhY2lkIjoiaNkMxN0Q4Mz E5QzqxQUMzRDM2QUZBRDY0Q0FFMDhBMigifQ==>

Для удобства использования написанного кода, он был разделен на заголовочный файл `Debit_getUserInfo_funcs.h` и файл исходного кода `Debit_getUserInfo_funcs.c`

Вышеопределенные user-story были успешно реализованы, используя функции библиотеки Sqlite: `sqlite3_open16`, `sqlite3_prepare`, `sqlite3_bind_***`, `sqlite3_step`, `sqlite3_close`.

Некоторые сложности были связаны с открытием базы данных, тк стандартная функция `sqlite3_open` принимает только UTF-8 параметр, в которой естественно не поддерживается русский текст. Просмотрев необходимую документацию, данная функция была заменена на аналогичную `sqlite3_open16`.

При помощи семейства функций `sqlite3_bind_***` удалось с легкостью передавать необходимые данные в `sqlite3_stmt`: `sqlite3_bind_int` - для целочисленных переменных, `sqlite3_bind_text` - для переменных типа `char*`, `sqlite3_bind_double` - для переменных с плавающей точкой.

В качестве примера можно продемонстрировать выполнение запроса на получение баланса. Сам запрос выглядит следующим образом:

```
const char *selectBalance="select balance from Account where client_id=? and  
account_id=?";
```

Функция, реализующая данный запрос:

```
if(sqlite3_prepare(db, selectBalance, -1, &statement, 0)!= SQLITE_OK)  
{  
    printf("Could not prepare statement: %s\n", sqlite3_errmsg(db));  
    return -1;  
}  
  
if (sqlite3_bind_int(statement, 1, clientID))  
{  
    printf("Could not bind integer: %s\n", sqlite3_errmsg(db));  
    return -1;  
}  
  
if (sqlite3_bind_int(statement, 2, accountID))  
{  
    printf("Could not bind integer: %s\n", sqlite3_errmsg(db));  
    return -1;  
}  
  
exitCode=sqlite3_step(statement);  
  
if(exitCode==SQLITE_DONE)  
{  
    printf("Wrong accountId or clientId!\n");
```

```

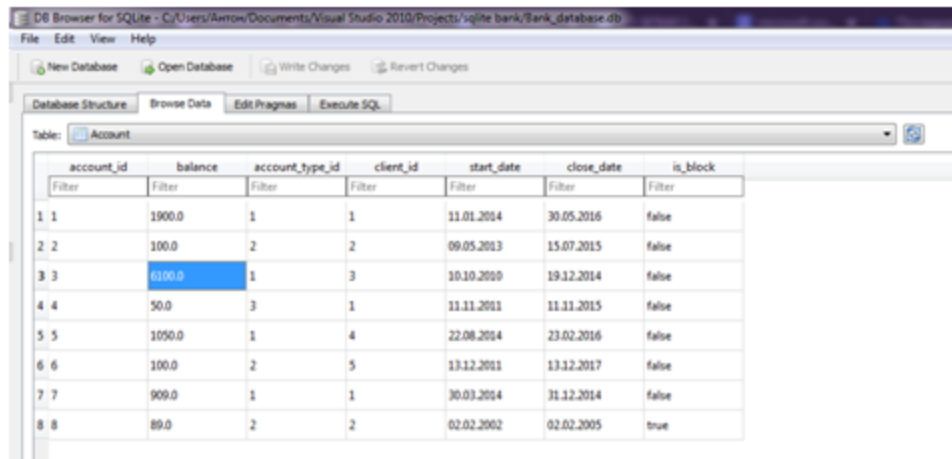
        return -1;
    }
    else if(exitCode==SQLITE_ROW)
        balance=sqlite3_column_double(statement, 0);

```

Как видно из приведенного кода, данные вытягиваются из `sqlite3_stmt` при помощи функций `sqlite3_column_***`.

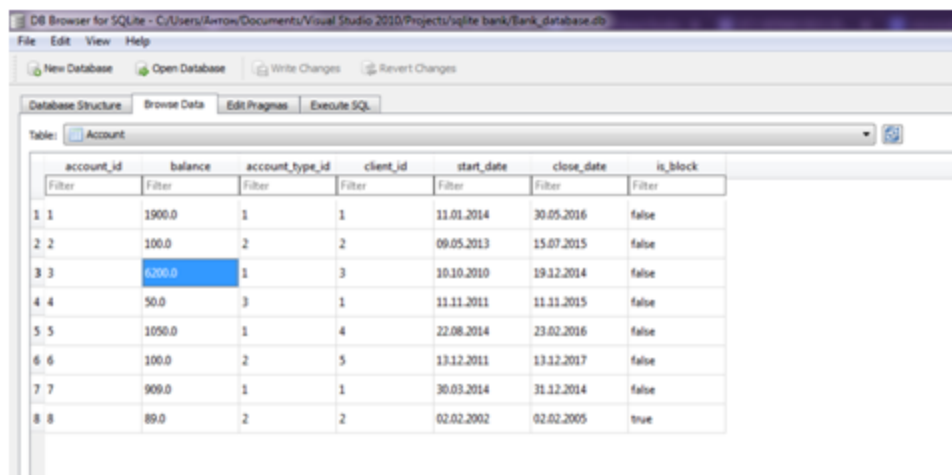
Для ведения истории пользовательских транзакций используется журнал, который содержит дату транзакции, её тип и номер пользовательского счета. Сначала необходимо убедиться, что в базе данных содержится тип транзакции `debit`. Если данного типа не существует, то производится вставка, если же существует - то нет.

Результат выполнения `Debit 100 USD user-story` приведен на следующих скриншотах. До выполнения:



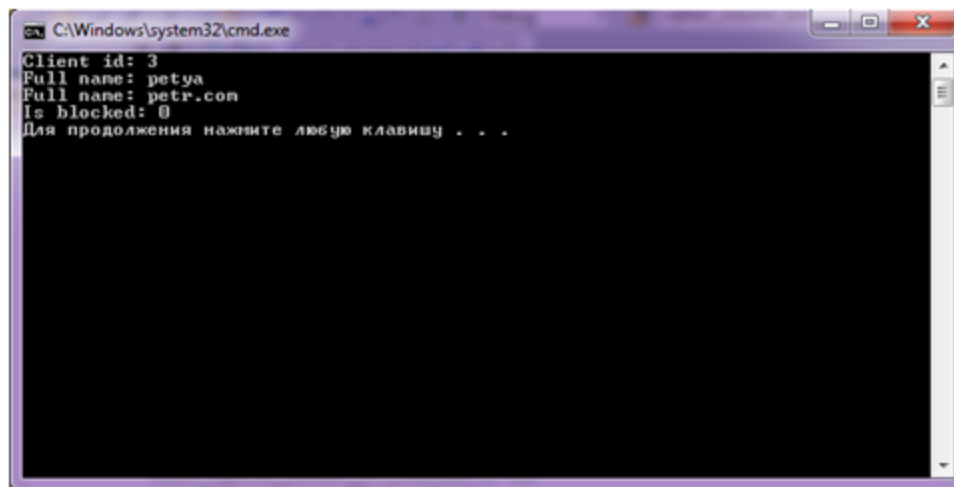
account_id	balance	account_type_id	client_id	start_date	close_date	is_block
1	1900.0	1	1	11.01.2014	30.05.2016	false
2	100.0	2	2	09.05.2013	15.07.2015	false
3	6100.0	1	3	10.10.2010	19.12.2014	false
4	50.0	3	1	11.11.2011	11.11.2015	false
5	1050.0	1	4	22.08.2014	23.02.2016	false
6	100.0	2	5	13.12.2011	13.12.2017	false
7	909.0	1	1	30.03.2014	31.12.2014	false
8	89.0	2	2	02.02.2002	02.02.2005	true

После выполнения:



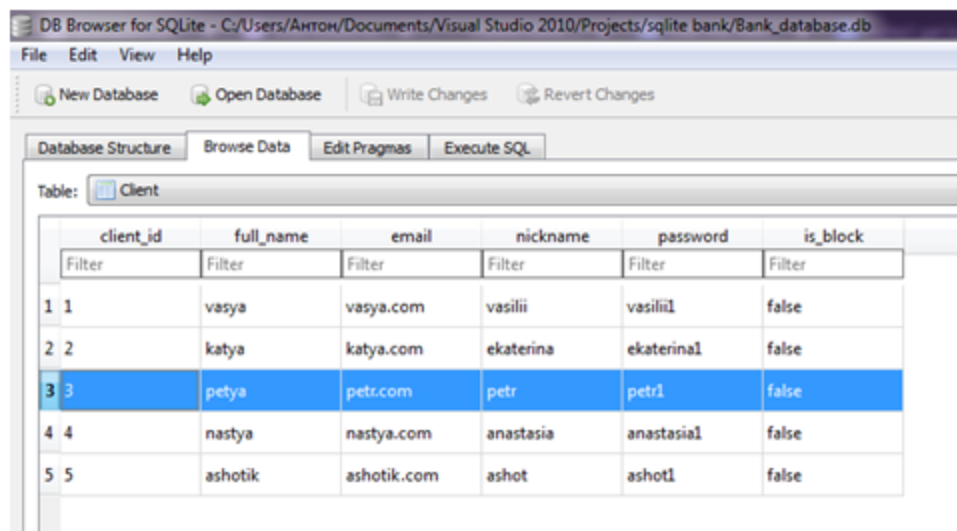
account_id	balance	account_type_id	client_id	start_date	close_date	is_block
1	1900.0	1	1	11.01.2014	30.05.2016	false
2	100.0	2	2	09.05.2013	15.07.2015	false
3	6200.0	1	3	10.10.2010	19.12.2014	false
4	50.0	3	1	11.11.2011	11.11.2015	false
5	1050.0	1	4	22.08.2014	23.02.2016	false
6	100.0	2	5	13.12.2011	13.12.2017	false
7	909.0	1	1	30.03.2014	31.12.2014	false
8	89.0	2	2	02.02.2002	02.02.2005	true

Первая из двух user-story(about client info), реализована с использованием все тех же функций, что и вторая. Результаты работы для клиента с id=3:



```
C:\Windows\system32\cmd.exe
Client id: 3
Full name: petya
Full name: petr.com
Is blocked: 0
Для продолжения нажмите любую клавишу . . .
```

Данные из sqlite browser-a:



DB Browser for SQLite - C:/Users/Антон/Documents/Visual Studio 2010/Projects/sqlite bank/Bank_database.db						
File Edit View Help						
New Database Open Database Write Changes Revert Changes						
Database Structure Browse Data Edit Pragma Execute SQL						
Table: Client						
	client_id	full_name	email	nickname	password	is_block
	Filter	Filter	Filter	Filter	Filter	Filter
1	1	vasya	vasya.com	vasili	vasili1	false
2	2	katya	katya.com	ekaterina	ekaterina1	false
3	3	petya	petr.com	petr	petr1	false
4	4	nastya	nastya.com	anastasia	anastasia1	false
5	5	ashotik	ashotik.com	ashot	ashot1	false

Захарова Анна.

1. as an administrator i can block user
2. as an administrator i can unblock user
3. as an administrator i can check operation history

Используемые параметры функций:

- sqlite3 *db - дескриптор БД;
- account_id - значение, идентифицирующее счет пользователя.

При реализации использовались следующие функции библиотеки sqlite3:

- sqlite3_prepare_v2() - подготовленный запрос;
- sqlite3_bind_*() - установление значений;
- sqlite3_step() - выполнение запроса и получение результата;

- `sqlite3_column_text()` - выборка значений.

Для реализации первой user story написана функция, имеющая следующую сигнатуру:

```
blockAccountByAccountID(sqlite3 *db, int account_id);
```

В функции используется запрос следующего вида:

```
char *sqlBlockAccountByAccountID = "UPDATE Account SET is_block = 1 WHERE  
account_id = ?";
```

Как видно, флаг блокировки счета устанавливается в 1.

Для реализации второй user story написана аналогичная функция

```
unblockAccountByAccountID(sqlite3 *db, int account_id);
```

С использованием запроса

```
char *sqlBlockAccountByAccountID = "UPDATE Account SET is_block = 0 WHERE  
account_id = ?";
```

Для проверки текущего состояния блокировки можно использовать функцию

```
int isAccountBlock(sqlite3 *db, int account_id);
```

Для реализации третьей user story используется функция

```
void getHistoryByAccountID(sqlite3 *db, int account_id);
```

Запрос к БД выглядит как показано ниже

```
char* sqlGetHistoryByAccountID = "SELECT L.log_date , O.operation_name FROM Log L  
INNER JOIN Operation O ON L.operation_id = O.operation_id where L.account_id = ?";
```

Результат работы метода при id счета = 1

```
date : '2014-11-21' operation : Deposit money  
date : '2014-11-22' operation : Transfer money
```

Рымарчик Александр.

Мне необходимо было реализовать следующие User-story:

- As an administrator i can delete user.
- As an administrator i can restore user.

Реализация данных функций располагается в файле исходного кода deleteAndRestoreClientByClientID.c по адресу

https://github.com/vitalbit/bank_app/tree/master/include. Смысл обеих функций состоит в задании и снятии флага с поля is_delete таблицы Client.

Для первой User-story использовалась функция deleteClientByClientID, принимающая на вход параметры sqlite3 *db - дескриптора БД и client_id - значения, идентифицирующего пользователя. В функции используется запрос следующего вида:

```
char *sqlDeleteClientByClientID = "UPDATE Client SET is_delete = 1 WHERE client_id = ?";
```

Как видно, флаг удаления клиента устанавливается в 1.

Обратное действие происходит во второй User-story, где использовалась функция restoreClientByClientID, принимающая на вход те же параметры. Тут запрос имеет следующий вид:

```
char *sqlRestoreClientByClientID = "UPDATE Client SET is_delete = 0 WHERE client_id = ?";
```

Флаг удаления клиента устанавливается в 0.

Вышеопределенные User-story были реализованы, используя функции библиотеки Sqlite: sqlite3_prepare_v2, sqlite3_bind_***, sqlite3_step и sqlite3_finalize.

Ввиду того, что качестве параметра в обоих случаях выступал client_id, имеющий целочисленное значение, мной использовалась функция sqlite3_bind_int.

Пример работы:

#	client_id	full_name	email	nickname	password	is_block	is_delete
1	1	Kyrganovich Vladislav	klass@tut.by	Vladislav	4452	0	0
2	2	Volkova Olga	volanchik@gmail.com	Volanchik	3214	0	0
3	3	Protko Ilona	ilonaTheBest@mail.com	Protko	9201	0	0
4	4	Dubova Darja	dubova12@mail.com	Dubik	7723	0	0
5	5	Bushik Marina	marinochka82@gmail.com	Marinochka	2184	0	0
6	6	Medvedeva Kristina	kristy007@gmail.com	Medved	1314	0	0
7	7	⌘	⌘	⌘	⌘	0	0

```
Enter client id:
6
Client was marked as deleted!
```

#	client_id	full_name	email	nickname	password	is_block	is_delete
1	1	Kyrganovich Vladislav	klass@tut.by	Vladislav	4452	0	0
2	2	Volkova Olga	volanchik@gmail.com	Volanchik	3214	0	0
3	3	Protko Ilona	ilonaTheBest@mail.com	Protko	9201	0	0
4	4	Dubova Darja	dubova12@mail.com	Dubik	7723	0	0
5	5	Bushik Marina	marinotchka82@gmail.com	Marinotchka	2184	0	0
6	6	Medvedeva Kristina	kristy007@gmail.com	Medved	1314	0	1
7	7	⌘	⌘	⌘	⌘	0	0

```
Enter client id:
6
Client was restored!
```

#	client_id	full_name	email	nickname	password	is_block	is_delete
1	1	Kyrganovich Vladislav	klass@tut.by	Vladislav	4452	0	0
2	2	Volkova Olga	volanchik@gmail.com	Volanchik	3214	0	0
3	3	Protko Ilona	ilonaTheBest@mail.com	Protko	9201	0	0
4	4	Dubova Darja	dubova12@mail.com	Dubik	7723	0	0
5	5	Bushik Marina	marinotchka82@gmail.com	Marinotchka	2184	0	0
6	6	Medvedeva Kristina	kristy007@gmail.com	Medved	1314	0	0
7	7	⌘	⌘	⌘	⌘	0	0

Литвинов Владимир.

1) Добавление нового пользователя.

Была создана новая команда, позволяющая создать нового пользователя в системе.

Сначала получают вводимые с клавиатуры пользовательские данные, потом

происходит валидация, нет ли пользователя с эквивалентным логином в системе.

Если нет тогда мы создаем нового пользователя, иначе выдаем сообщение об ошибке.

Запросы:

"select client_id from Client where nickname=?"

**"insert into Client(full_name, email, nickname, password, is_block, is_delete)
values(?,?,?,?,?,?)"**

Подстановка аргументов и выполнение запросов осуществляются аналогично с уже неоднократно описанным выше методом.

2) Проверка баланса.

Получаем с клавиатуры id счета, и выполняем запрос.

"select balance from Account where account_id=?"

В случае получения пустого результата выводим соответствующее сообщение.

Пример работы:

```

Enter new client full name <first and last names>:
1 1
Enter new client email:
1234
Enter new client nickname:
124
Enter new client password:
123
Client created.
1. See all account.
2. Credit money.
3. Block account
4. View the history of user operations
5. Unblock account
6. Delete client <by client id>
7. Debit money
8. Get user info
9. Create client
10. Check balance
11. Edit client information
12. Add account to client
10
Enter client's account id:
1
Balance = 790000.000000
1. See all account

```

SELECT * FROM Client						Query
client_id	full_name	email	nickname	password	is_block	is_delete
1	Kyrganovich Vl...	vlad92@gmail.c...	Vladislav	4452	false	false
2	Volkova Olga	volanchik@gm...	Volanchik	3214	false	false
3	Protko Ilona	ilonaTheBest@...	Protko	9201	false	false
4	Dubova Darja	dubova12@mai...	Dubik	7723	false	false
5	Bushik Marina	marinochka82...	Marinochka	2184	false	false
6	Medvedeva Kris...	kristy007@gmai...	Medved	1314	false	false
7	W L	123	123	123	false	false
8	1 1	1234	124	123	false	false

Алтыев Мурад.

- As an administrator I can delete user account

Для реализации первой user-story была создана функция:

```
void deleteAccountByClientId(sqlite3 *db, int client_id);
```

В качестве параметров передаются дескриптор базы и id клиента, вводимый с клавиатуры. Затем из базы удаляется аккаунт клиента с помощью следующего запроса:

```
delete from Account where client_id=?;
```

Исходный код данной функции находится в файле deleteAccountByClientId.c, расположенного по адресу: https://github.com/vitalbit/bank_app/tree/master/include


Пример работы:

```

Enter client id: 2
Account has been deleted.


```

Данные из sqlite browser'a:

Table: Account  New Record Delete Record

	account id	balance	account type	ic	client id	start date	close date	is bloc
1	1	792008.0	1	1	1	2010-08-12		
2	2	13300000.0	1	2	2	2012-02-21		
3	3	2000000.0	2	1	1	2011-01-12		
4	4	9002000.0	3	4	4	2000-04-11		
5	5	210000.0	2	5	5	2001-05-10		
6	6	12.0	1	1	1	2014-12-14		

До выполнения

Table: Account  New Record Delete Record

	account id	balance	account type	ic	client id	start date	close date	is bloc
1	1	792008.0	1	1	1	2010-08-12		
2	3	2000000.0	2	1	1	2011-01-12		
3	4	9002000.0	3	4	4	2000-04-11		
4	5	210000.0	2	5	5	2001-05-10		
5	6	12.0	1	1	1	2014-12-14		

После выполнения

- As an administrator I can check block on client

Для реализации второй user-story была создана функция:

void checkBlockOnClient(sqlite3 *db, int client_id);

В качестве параметров также передаются дескриптор базы и id клиента. Функция возвращает на экран информацию о том, заблокирован ли клиент (в базе: 0 - не заблокирован, 1 - заблокирован).

Запрос:

select is_block from Client where client_id=?;


Исходный код данной функции также расположен по адресу:

https://github.com/vitalbit/bank_app/tree/master/include в файле checkBlockOnClient.c.

Выполнение работы:

```
Enter client id: 3
Client is not blocked
```

Данные из sqlite browser'a:

Table: Client  New Record Delete Record

	client id	full name	email	nickname	password	is block	is dele
1	1	Kyrganovich Vladislav	klass@tut.by	Vladislav	4452	0	
2	2	Volkova Olga	volanchik@gmail.com	Volanchik	3214	0	
3	3	Protko Iлона	ilonaTheBest@mail.ru	Protko	9201	0	
4	4	Dubova Darja	dubova12@mail.ru	Dubik	7723	0	
5	5	Bushik Marina	marinochka82@gmail.com	Marinochka	2184	0	
6	6	Medvedeva Kristina	kristy007@gmail.com	Medved	1314	0	
7	7	г	г	г	г	0	