Содержание

[Введение 2](#_Toc468382473)

[Глава 1. Описание проекта 3](#_Toc468382474)

[1.1. Краткий обзор альтернативных решений 3](#_Toc468382475)

[1.2. Основной функционал приложения 7](#_Toc468382476)

[1.3. Описание предметной области 8](#_Toc468382477)

[1.4. Описание алгоритма синхронизации 10](#_Toc468382478)

[1.5. Выбор средств разработки 12](#_Toc468382479)

[Глава 2 . Реализация проекта 13](#_Toc468382480)

[2.1. Основные требования к оборудованию и программному обеспечению 13](#_Toc468382481)

[2.2. Описание организации данных 14](#_Toc468382482)

[2.3. Описание основных запросов 22](#_Toc468382483)

[2.4. Алгоритм работы синхронизации данных. 25](#_Toc468382484)

[Заключение 31](#_Toc468382485)

[Список литературы 32](#_Toc468382486)

# Введение

Ведение и планирование финансового бюджета всегда являлось важной частью процесса экономии. Нередко люди, не задумываясь, покупают лишние товары в магазинах или даже не знают, что в другом магазине можно купить дешевле.

В современном мире в веке информационных технологий отличным вариантом для контроля бюджета была бы соответствующая программа, доступная на ПК, планшете и телефоне, позволяющая легко и быстро синхронизировать данные с помощью сети Интернет. Программа, представляющая бюджет в наглядном виде и подсказывающая, на каких тратах в будущем пользователи могли бы сэкономить.

Таким образом, цель работы – разработать приложение для управления личными финансами.

Задачи работы:

1. Цуйцуй
2. Йцйцуй
3. 3414

# Глава 1. Описание проекта

## Краткий обзор альтернативных решений

Перед тем, как приступить к разработке приложения для ведения и планирования бюджета, был произведен поиск и анализ аналогичных приложений.

Основной функционал всех рассмотренных решений – создание счетов и перевод средств между ними, регистрация и планирование расходов и доходов с уведомлениями.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Название | Достоинства | Недостатки | Бесплатная версия | Цена |
| Домашняя бухгалтерия | - многопользовательская система;  - импорт данных из XLS (Microsoft Excel), QIF (Quicken Interchange Format) и CSV файлов;  - экспорт данных в 15 различных форматов;  - учёт денег отданных и взятых в долг;  - возможность использовать до 5-ти валют включительно;  - обмен валют. | - нет уведомлений о предстоящих расходах;  - синхронизация не прозрачна для пользователя;  - бесплатную версию нужно зарегистрировать;  - только для Windows OS. | Либо 30-дневная, либо очень ограниченная в возможностях версия (только регистрация доходов и расходов). | 500-700 руб. |
| MoneyTracker | - резервное копирование;  - многопользовательская система. | - нет синхронизации;  - непродуманный интерфейс;  - нет защиты данных паролем;  - только для Windows OS. | Бесплатная версия ограничена по количеству данных | 500-800 руб. |
| Family 11 | - синхронизация через удалённый сервер;  - возможность ставить цели. | - нет уведомлений;  - хранит данные в открытом виде;  - во время работы программы возникают ошибки;  - только для Windows OS. | 30-дневный пробный период. | 5-20$ (350-1000 руб.) |
| Домашние финансы | - сохранение отчётов в XLS (excel);  - управление базами данных. | - устаревший интерфейс;  - нет уведомлений;  - хранит данные в открытом виде;  - отсутствие какой-либо справочной системы;  - только для Windows OS. | Есть бесплатная версия | 400-4000 руб. |
| AceMoney | - есть версия для Mac OS;  - архивирование;  - импорт и экспорт в форматы txt, csv, qif, xml;  - резервное копирование и восстановление. | - во время работы программы возникают ошибки;  - синхронизация не работает;  - нет уведомлений;  - рассчитана на одного пользователя;  - только для Windows OS. | Возможна работа только с 1 счётом | 600 руб. |
| Свои деньги | - резервное копирование и восстановление;  - справочная система;  - экранная лупа;  - курсы валют из интернета. | - рассчитана на одного пользователя;  - «Мастер первого запуска», заставляющий заполнять различные данные;  - сложное ведение счёта;  - непонятный интерфейс;  - нет уведомлений;  - нет синхронизации;  - только для Windows OS. | Полностью бесплатна. | 0 руб. |
| GnuCash | - кроссплатформенная. | - тяжеловесная;  - сложная в освоении;  - нет планирования;  - нет синхронизации;  - ориентирована на людей, знакомых с бухгалтерским учётом. | Полностью бесплатна. | 0 руб. |

Таким образом, основные недостатки рассмотренных программ:

- нет уведомлений о запланированных тратах;

- нет синхронизации или она не работает;

- недостаточная защита данных;

- большинство программ только для Windows OS;

- большинство программ для одного пользователя.

## Основной функционал приложения

Исходя из проведённого анализа, можно сформулировать основной функционал будущего приложения для ведения и планирования бюджета, а именно следующие пункты:

* Сохранение данных о счетах, доходах и расходах с возможностью их корректировать и удалять.
* Перевод средств между счетами.
* Планирование будущих расходов и доходов с возможностью указать периодичность.
* Уведомления о запланированных операциях. Например: «Завтра с вашего счёта будет списано 90 рублей на оплату телефона МТС».
* Синхронизация данных пользователя приложения (на разных устройствах) через централизованный сервер.
* Авторизация и аутентификация пользователей. Шифрование паролей.
* Резервное копирование и восстановление данных.
* Возможность устанавливать, изменять и добавлять категории, единицы измерения, валюты.
* Построение статистических отчётов в виде диаграмм.

## Описание предметной области

Контроль расходов и доходов является основной составляющей ведения домашних финансов. Статистика свидетельствует, что простой учёт трат позволяет сократить расходы на 20%.

Цель создаваемого приложения – помочь людям вести свой личный бюджет, и, основываясь на получаемых данных, давать рекомендации по экономии средств.

Разрабатываемое приложение будет использоваться для ведения и планирования бюджета в домашних условиях. Приложение будет иметь возможность создания одного или нескольких счетов, регистрации расходов и доходов с возможностью планирования, возможность синхронизации с другими устройствами. Приложение будет многопользовательским и кроссплатформенным.

В приложении будут присутствовать следующие бизнес-процессы:

1. Планирование бюджета – пользователь сможет запланировать доходы и расходы, а так же будет иметь возможность сделать их периодическими;
2. Создание отчётности – пользователь сможет видеть автоматически создаваемые отчёты по бюджету;
3. Синхронизация с удалённым сервером – пользователь будет иметь возможность синхронизировать данные, хранимые локально, с удалённым сервером, благодаря чему пользователь сможет получать данные с различных устройств;
4. Анализ расходов – программа будет автоматически анализировать расходы (или запланированные расходы) пользователя и предлагать варианты для экономии.

Ниже представлены фрагменты модели предметной области в виде диаграммы классов:

1. Рисунок 1. Фрагмент, отображающий обработку доходов и расходов.
2. Рисунок 2. Фрагмент, отображающий обработку запланированных операций.

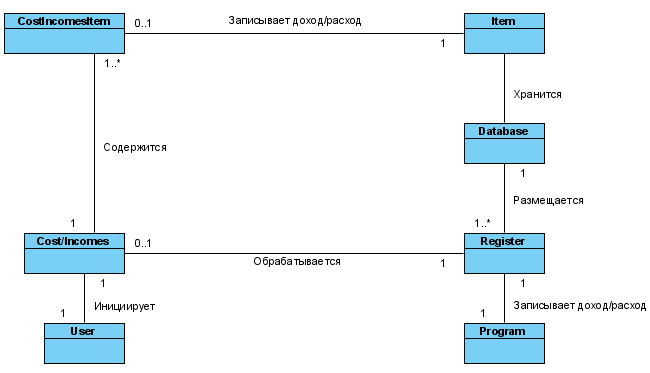


Рисунок Фрагмент модели предметной области, отображающий обработку доходов и расходов.

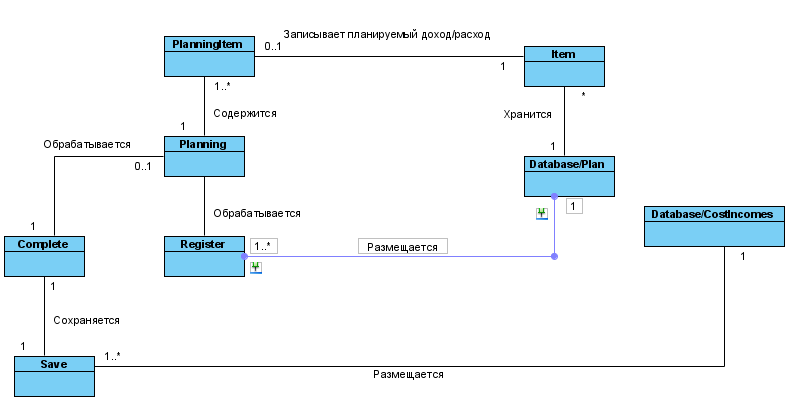


Рисунок Фрагмент модели предметной области, отображающий обработку запланированных операций.

## Описание алгоритма синхронизации

Синхронизация осуществляется между локальной базой данных и базой, установленной на удалённом сервере.

Репликация баз данных в данном случае не подходит, т.к. база клиента отличается по структуре от базы сервера. На базе сервера хранятся данные всех пользователей программы. На базе клиента будут храниться только данные пользователей, работающих на данном ПК.

Синхронизация проводится с помощью удалённого сервера. При запуске синхронизации программа соединяется с удалённым сервером, сравнивает свою базу данных с удалённой версией и загружает различия себе либо в удалённую базу. Такой способ обеспечивает прозрачность процесса для пользователя.

Синхронизация реализована с помощью технологии WebSockets.

При разработке алгоритма возникла сложность с удаляемыми записями: поскольку необходимо гарантировать целостность данных, то невозможно просто удалять данные с клиента, а затем с сервера. Потому что в таком случае возможна ситуация, когда другой клиент запишет в базу сервера данные, которые были уже удалены первым клиентом.

Для решения этой проблемы был разработан следующий алгоритм:

1. Каждое устройство при синхронизации передаёт свой уникальный ID (UUID устройства). Он записывается в соответствующую таблицу на сервере, туда же записывается уникальный ID пользователя (UUID пользователя). Соответственно, один пользователь может синхронизироваться с разных устройств.
2. Так же на сервере создана таблица, хранящая информацию о записях, которые были удалены с устройств. Структура таблицы: UUID устройства, UUID пользователя, UUID записи и имя таблицы, из которой запись удалена.
3. Если синхронизация ещё ни разу не проводилась, то записи на устройстве удаляются как обычно. Если же синхронизация уже была, то работает следующий алгоритм:
   1. Когда пользователь «удаляет» запись на клиенте, запись на самом деле не удаляется, но у неё обновляется поле DeletedDate.
   2. При запуске синхронизации все записи с обновлённым полем DeletedDate передаются серверу.
   3. Сервер проверяет по таблице DeletedNotes сколько устройств так же удалили запись. Если их столько же, сколько всего устройств у данного пользователя (в таблице Devices), то это значит, что запись удалена со всех устройств. В таком случае запись удаляется и из базы на сервере, и из базы на клиенте, который передал запрос. Соответственно, из таблицы DeletedNotes запись так же удаляется.
   4. Если ещё не все устройства удалили запись, то сервер добавляет её в таблицу DeletedNotes и возвращает клиенту дату последней синхронизации.

## Выбор средств разработки

Для разработки данного приложения был выбран язык Java 1.8. Это современный объектно-ориентированный язык, имеющий строгую типизацию данных, что уменьшает вероятность ошибок. Кроме того, приложение, написанное на Java, будет кроссплатформенным.

Для создания интерфейса приложения был использован Java Scene Builder, а сам интерфейс построен с использованием современной библиотеки JavaFX.

База данных для локальной версии приложения выбиралась по простоте установки и полноте функциональности.

В качестве базы данных для локальной версии приложения была выбрана SQLite, а для серверной базы данных – СУБД MySQL.

Для реализации синхронизации с сервером была выбрана технология WebSockets – асинхронный протокол обмена данными посредством http-запросов.

Для сборки проекта был выбран фреймворк Maven.

Для передачи сообщений посредством json и их последующей расшифровки были использованы библиотеки JSON Simple и Jackson.

# Глава 2 . Реализация проекта

## Основные требования к оборудованию и программному обеспечению

* минимальные системные требования:
* процессор – Intel Core 2 Dual 2MHZ;
* оперативная память - 1GB RAM;
* жесткий диск - 5 GB свободной памяти жесткого диска;
* видео – SVGA 800 x 600;
* операционная система: Windows XP и выше, либо с ОС Linux c веб сервером.
* языки и среды:
* Java 1.8 JRE;

## Описание организации данных

Так как потребуется хранить большие объёмы данных, в приложении используется СУБД SQLite и MySQL на сервере. Требуется создать две базы данных – HomeAccounting в приложении и RemoteAccounting на сервере. БД HomeAccounting содержит 10 таблиц: Bills, Categories, Currencies, Measures, MoneyTurn, Planning, TimeIntervals, Users, Cashes, LastSyncTime.

Краткое описание таблиц:

Bills – таблица счетов, содержит полную информацию о счетах пользователя, включает в себя поля:

- BillId – ключевое поле;

- BillName – название счёта;

- Date – дата создания счёта;

- StartBalance – начальный баланс;

- Note – заметки;

- UserId – ID пользователя счёта;

- UUIDBills – глобально-уникальный идентификатор записи;

- UpdatedDate – дата последнего обновления записи;

- DeletedDate – дата удаления записи;

- InsertDate – дата добавления записи;

Categories – таблица, хранящая категории, содержит поля:

- CategoryId – ключевое поле;

- CategoryName – название категории;

- CostIncome – является категория статьёй расходов или доходов.

- UUIDCategory – глобально-уникальный идентификатор записи;

- UpdatedDate – дата последнего обновления записи;

- DeletedDate – дата удаления записи;

- InsertDate – дата добавления записи;

Currencies – таблица, хранящая валюты, содержит в себе поля:

- CurrencyId – ключевое поле;

- CurrencyName – название валюты;

- IsMain – является ли валюта основной для счёта.

- NameShort – короткое название.

- UUIDCurrency – глобально-уникальный идентификатор записи;

- UpdatedDate – дата последнего обновления записи;

- DeletedDate – дата удаления записи;

- InsertDate – дата добавления записи;

Measures – таблица, хранящая единицы измерения, содержит поля:

- MeasureId – ключевое поле;

- MeasureName – единица измерения;

- UUIDMeasure – глобально-уникальный идентификатор записи;

- UpdatedDate – дата последнего обновления записи;

- DeletedDate – дата удаления записи;

- InsertDate – дата добавления записи;

MoneyTurn – таблица для сохранения доходов и расходов, содержит поля:

- MoneyTurnId – ключевое поле;

- Date – дата создания записи;

- BillId – счёт, связанный с записью;

- Count – количество приобретённого товара или проведённых операций;

- MeasureId – единица измерения, связанная с записью;

- Note – примечание;

- CostIncome – запись является доходом или расходом;

- Amount – потраченная или приобретённая сумма;

- CategoryId – категория, связанная с записью;

- UUIDMoneyTurn – глобально-уникальный идентификатор записи;

- UpdatedDate – дата последнего обновления записи;

- DeletedDate – дата удаления записи;

- InsertDate – дата добавления записи;

Planning – таблица, регистрирующая запланированные доходы и расходы, содержит поля:

- PlanId – ключевое поле;

- Date – дата запланированного учёта;

- BillId – счёт для запланированного учёта;

- TimeIntervalId – периодичность;

- Count – количество приобретаемого товара или проведённых операций;

- MeasureId – единица измерения;

- Note – примечание;

- Amount – сумма;

- CostIncome – доход или расход;

- CategoryId – категория;

- UUIDPlan – глобально-уникальный идентификатор записи;

- UpdatedDate – дата последнего обновления записи;

- DeletedDate – дата удаления записи;

- InsertDate – дата добавления записи;

TimeIntervals – таблица, хранящая варианты временных интервалов, содержит поля:

- TimeIntervalId – ключевое поле;

- TimeIntervalName – временной интервал;

- UUIDTimeInterval – глобально-уникальный идентификатор записи;

- UpdatedDate – дата последнего обновления записи;

- DeletedDate – дата удаления записи;

- InsertDate – дата добавления записи.

Users – таблица, хранящая информацию о пользователе, содержит поля:

- UserId – ключевое поле;

- Username – имя пользователя;

- Password – пароль в зашифрованном виде;

- E-mail – электронный адрес пользователя.

- UUIDUser – глобально-уникальный идентификатор пользователя;

- UpdatedDate – дата последнего обновления записи;

- DeletedDate – дата удаления записи;

- InsertDate – дата добавления записи;

LastSyncTime – таблица, хранящая дату последней синхронизации, содержит поля:

- LastSyncTimestamp – дата последней синхронизации.

Ниже представлена ER-диаграмма данной базы (см. рисунок 1).

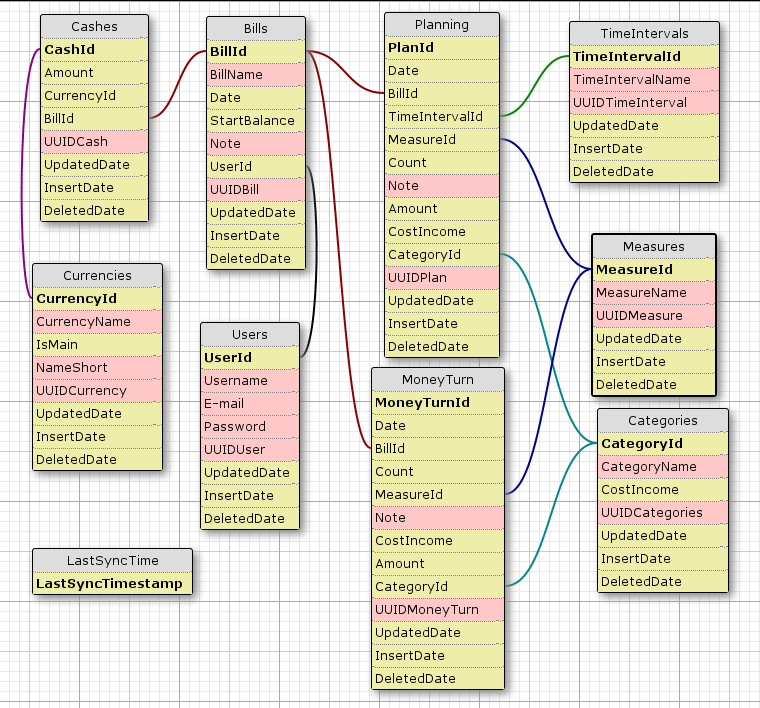


Рис. 1. ER-диаграмма

БД RemoteAccounting содержит 12 таблиц: Bills, Categories, Currencies, Measures, MoneyTurn, Planning, TimeIntervals, Users, Cashes, LastSyncTime, Devices, DeletedNotes.

Краткое описание таблиц:

Bills – таблица счетов, содержит полную информацию о счетах пользователя, включает в себя поля:

- UUIDBills – глобально-уникальный идентификатор записи;

- BillName – название счёта;

- Date – дата создания счёта;

- StartBalance – начальный баланс;

- Note – заметки;

- UserId – ID пользователя счёта;

- UpdatedDate – дата последнего обновления записи;

- DeletedDate – дата удаления записи;

- InsertDate – дата добавления записи;

Categories – таблица, хранящая категории, содержит поля:

- UUIDCategory – глобально-уникальный идентификатор записи;

- CategoryName – название категории;

- CostIncome – является категория статьёй расходов или доходов.

- UpdatedDate – дата последнего обновления записи;

- DeletedDate – дата удаления записи;

- InsertDate – дата добавления записи;

Currencies – таблица, хранящая валюты, содержит в себе поля:

- UUIDCurrency – глобально-уникальный идентификатор записи;

- CurrencyName – название валюты;

- IsMain – является ли валюта основной для счёта.

- NameShort – короткое название.

- UpdatedDate – дата последнего обновления записи;

- DeletedDate – дата удаления записи;

- InsertDate – дата добавления записи;

Measures – таблица, хранящая единицы измерения, содержит поля:

- UUIDMeasure – глобально-уникальный идентификатор записи;

- MeasureName – единица измерения;

- UpdatedDate – дата последнего обновления записи;

- DeletedDate – дата удаления записи;

- InsertDate – дата добавления записи;

MoneyTurn – таблица для хранения доходов и расходов, содержит поля:

- UUIDMoneyTurn – глобально-уникальный идентификатор записи;

- Date – дата создания записи;

- BillId – счёт, связанный с записью;

- Count – количество приобретённого товара или проведённых операций;

- MeasureId – единица измерения, связанная с записью;

- Note – примечание;

- CostIncome – запись является доходом или расходом;

- Amount – потраченная или приобретённая сумма;

- CategoryId – категория, связанная с записью;

- UpdatedDate – дата последнего обновления записи;

- DeletedDate – дата удаления записи;

- InsertDate – дата добавления записи;

Planning – таблица, регистрирующая запланированные доходы и расходы, содержит поля:

- UUIDPlan – глобально-уникальный идентификатор записи;

- Date – дата запланированного учёта;

- BillId – счёт для запланированного учёта;

- TimeIntervalId – периодичность;

- Count – количество приобретаемого товара или проведённых операций;

- MeasureId – единица измерения;

- Note – примечание;

- Amount – сумма;

- CostIncome – доход или расход;

- CategoryId – категория;

- UpdatedDate – дата последнего обновления записи;

- DeletedDate – дата удаления записи;

- InsertDate – дата добавления записи;

TimeIntervals – таблица, хранящая варианты временных интервалов, содержит поля:

- UUIDTimeInterval – глобально-уникальный идентификатор записи;

- TimeIntervalName – временной интервал;

- UpdatedDate – дата последнего обновления записи;

- DeletedDate – дата удаления записи;

- InsertDate – дата добавления записи.

Users – таблица, хранящая информацию о пользователе, содержит поля:

- UUIDUser – глобально-уникальный идентификатор пользователя;

- Username – имя пользователя;

- Password – пароль в зашифрованном виде;

- E-mail – электронный адрес пользователя.

- UpdatedDate – дата последнего обновления записи;

- DeletedDate – дата удаления записи;

- InsertDate – дата добавления записи;

LastSyncTime – таблица, хранящая дату последней синхронизации, содержит поля:

- LastSyncTimestamp – дата последней синхронизации.

Devices – таблица для хранения устройств, на которых установлена программа-клиент одного пользователя.

- UUIDDevice – глобально-уникальный идентификатор пользователя;

- UUIDUser – идентификатор пользователя;

- Device – уникальное имя устройства.

DeletedNotes – таблица для временного хранения удалённых записей. Сохраняет данные об удалённых записях, пока они не будут удалены на всех устройствах.

* UUIDDevice – идентификатор устройства;
* UUIDNote – идентификатор записи;
* TableName – имя таблицы, из которой удалена запись.

Ниже представлена ER-диаграмма данной базы (см. рисунок 2).

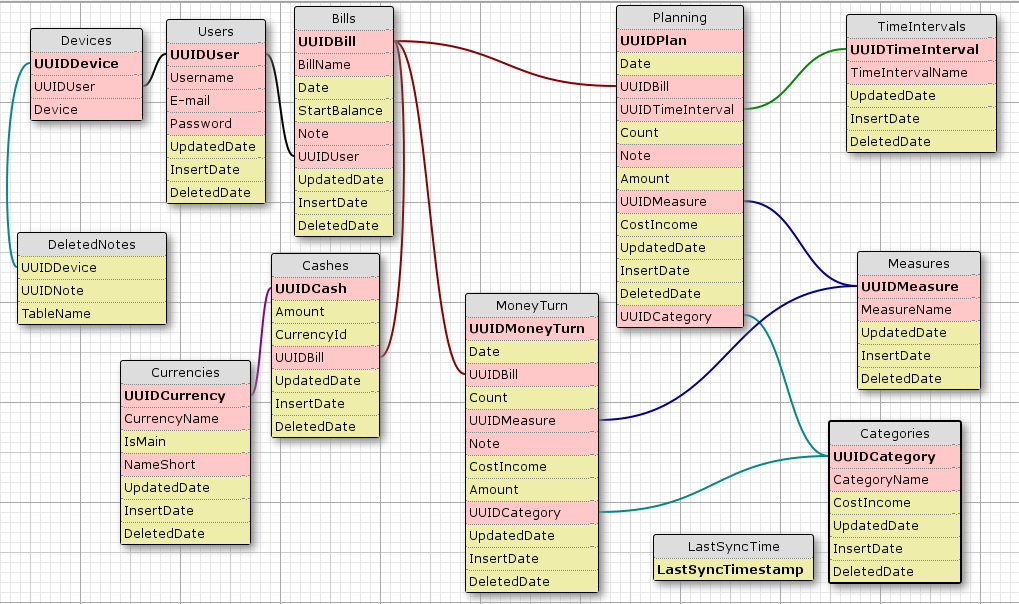


Рис. 2. ER-диаграмма

## Описание основных запросов

SQL является общепринятым языком запросов в реляционных базах данных. Для управления базами данных существуют специальные программы – системы управления базами данных (СУБД).

Каждый столбец представляет собой атрибут или совокупность атрибутов объектов. Часто в отношении столбца используется термин поле с указанием имени. Поле строки является минимальным элементом таблицы. Каждый столбец в таблице имеет определенное имя, тип данных и размер. Имена столбцов должны быть уникальны в пределах таблицы. Каждая строка представляет собой совокупность атрибутов конкретного объекта, например.

Одной из важнейших операций, которые выполняются при работе с данными, является выборка хранящейся в базе данных информации. Выборка представляет собой sql-запрос (query).

Для реализации приложения «Домашняя бухгалтерия» также использовались sql запросы. Ниже приведены запросы, используемые в программе:

* bills\_list;
* bills\_data;
* bills\_create;
* bills\_update;
* bills\_delete;
* currencies\_list;
* currencies\_create;
* currencies\_update;
* currencies\_delete;
* measures\_list;
* measures\_create;
* measures\_update;
* measures\_delete;
* timeintervals\_list;
* categories\_list;
* categories\_create;
* categories\_update;
* categories\_delete;
* moneyturn\_list;
* moneyturn\_create;
* moneyturn\_update;
* moneyturn\_delete;
* plan\_list;
* plan\_create;
* plan\_update;
* plan\_delete;
* user\_list;
* user\_data;
* user\_create;
* user\_update;
* user\_delete;
* user\_auth.

Рассмотрим несколько примеров запросов, используемых в приложении:

Запрос INSERT используется для создания новой записи в таблице. Для обновления уже существующих данных используется запрос UPDATE.

"INSERT Into MoneyTurn(Date, BillId, Count, MeasureId, CategoryId, Note, CostIncome, Amount) VALUES (" + date + "," + billId + "," + count + "," + measureId + "," + categoryId + "," + note + "," + costIncome + "," + amount + ")"

Данный запрос вставляет в таблицу MoneyTurn следующие значения:

- дата создания записи, Date;

- уникальный идентификатор счёта, BillId;

- количество приобретённого товара или совершённых операций, Count;

- уникальный идентификатор единиц измерения, MeasureId;

- уникальный идентификатор категории, CategoryId;

- примечания, Note;

- является запись доходом или расходом, CostIncome;

- сумма, Amount.

Для выборки информации из БД, используется запрос SELECT. Базовое действие этого запроса ограничено одной таблицей, хотя существуют конструкции, обеспечивающие выборку из нескольких таблиц одновременно.

"SELECT \* FROM categories WHERE CostIncome = " + costIncome"

Этот запрос выбирает из базы категории доходов или расходов (см. рисунок 3).

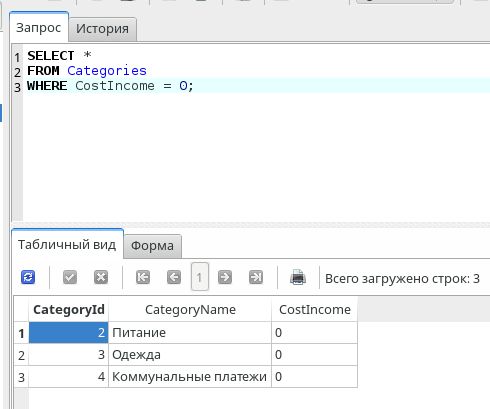


Рис. 3. Результат выполнения запроса

## Алгоритм работы синхронизации данных.

Для запуска синхронизации пользователь должен указать в настройках программы логин (имя пользователя) и пароль, используемые для авторизации на удалённом сервере. Если у пользователя нет учётной записи на сервере, то она будет создана. Если пользователь не указал данные настройки, то программа предложит ему это сделать. Далее алгоритм работает следующим образом:

1. Программа пытается соединиться с сервером.

private homeaccWSCClient tread;

public void connectToWebServer() {

try {

tread = new homeaccWSCClient(new URI("ws://localhost:8025/server/end"));

tread.connect();

} catch (URISyntaxException e) {

e.printStackTrace();

}

}

1. Когда соединение установлено, программа отправляет серверу все данные в виде json-сообщения. Сервер извлекает данные из полученного сообщения и записывает их в свою базу данных.

Так выглядят передаваемые json-сообщения:  
{"bills":[{"date":147497205600000,"billName":"weqweq","note":"wqe","startBalance":12323.0,"cashAmount":0.0,"userId":2,"billId":1,"billCurrency":null},{"date":1474934400000,"billName":"dsfs","note":"чсм","startBalance":10.0,"cashAmount":0.0,"userId":2,"billId":6,"billCurrency":null},{"date":1474934400000,"billName":"dfghjkl","note":null,"startBalance":10000.0,"cashAmount":0.0,"userId":2,"billId":7,"billCurrency":null},{"date":1474934400000,"billName":"ads","note":null,"startBalance":190.0,"cashAmount":0.0,"userId":2,"billId":8,"billCurrency":null},{"date":1474934400000,"billName":"dfsdf","note":null,"startBalance":340.0,"cashAmount":0.0,"userId":2,"billId":9,"billCurrency":null},{"date":1474934400000,"billName":"xcc","note":null,"startBalance":40.0,"cashAmount":0.0,"userId":2,"billId":10,"billCurrency":null}],"codeOperation":"bills","numSync":"first"}

{"measures":[{"measureId":1,"measureName":"шт","uuidMeasure":null,"deletedDate":0},{"measureId":2,"measureName":"десяток","uuidMeasure":null,"deletedDate":0},{"measureId":3,"measureName":"литр","uuidMeasure":null,"deletedDate":0},{"measureId":4,"measureName":"опс","uuidMeasure":null,"deletedDate":0}],"codeOperation":"measures","numSync":"first"}

1. После первой синхронизации сервер передаёт дату синхронизации и она сохраняется на клиенте.

{"value":1478995200000,"key":"date"}

public static void setLastSyncDate(long date) {

LocalDate oldDate = CommonDAO.getLastSyncDate();

if (oldDate == null) {

Connection c = DBapi.getConnection();

PreparedStatement ps = c.prepareStatement("INSERT INTO LastSyncTimestamp(LastSyncTime) VALUES (?)");

ps.setLong(1, date);

ps.executeUpdate();

ps.close();

} else {

Connection c = DBapi.getConnection();

PreparedStatement ps = c.prepareStatement("UPDATE LastSyncTimestamp SET LastSyncTime = ?");

ps.setLong(1, date);

ps.executeUpdate();

ps.close();

}

1. При последующих попытках синхронизации программа будет выбирать из своей базы только те данные, которые были изменены, добавлены или удалены после даты, присланной с сервера. После чего программа отправит эти данные серверу в виде json-сообщения, а сервер извлечёт их и запишет в свою базу данных.

Например, данные из таблицы Bills. Получение данных:

public static ObservableList<Bills> selectBillsSyncDataFromLastDate(LocalDate lastSyncDate) throws SQLException, ClassNotFoundException {

ResultSet rs;

try (Connection c = DBapi.getConnection();

PreparedStatement ps = c.prepareStatement("SELECT b.Date, b.BillName, b.Note, b.StartBalance, b.BillId, b.UserId, b.DeletedDate, b.UUIDBill FROM Bills b WHERE b.UpdatedDate BETWEEN ? AND (SELECT b.UpdatedDate From Bills b)");

) {

long longDate = LocalDateReciever.getLongTimeFromLocalDate(lastSyncDate);

ps.setLong(1, longDate);

rs = ps.executeQuery();

ObservableList<Bills> billsList = FXCollections.observableArrayList();

while ( rs.next() ) {

billsList.add(new Bills(rs.getLong("Date"), rs.getString("BillName"), rs.getString("Note"), rs.getDouble("StartBalance"), rs.getInt("BillId"), rs.getInt("UserId"), rs.getLong("DeletedDate"), rs.getString("UUIDBill")));

}

rs.close();

ps.close();

return billsList;

}

}

Отправка на сервер:

resultJson.forEach((temp) -> Syncronization.getInstance().sendMessage(temp));

Вид JSON:

{"bills":[{"date":1474934400000,"billName":"dsfs","note":"чсм","startBalance":10.0,"cashAmount":0.0,"userId":2,"billId":6,"uuidBill":"","uuidUser":null,"delDate":0,"billCurrency":null},{"date":1474934400000,"billName":"dfghjkl","note":null,"startBalance":10000.0,"cashAmount":0.0,"userId":2,"billId":7,"uuidBill":"","uuidUser":null,"delDate":0,"billCurrency":null},{"date":1474934400000,"billName":"ads","note":null,"startBalance":190.0,"cashAmount":0.0,"userId":2,"billId":8,"uuidBill":"","uuidUser":null,"delDate":0,"billCurrency":null},{"date":1474934400000,"billName":"dfsdf","note":null,"startBalance":340.0,"cashAmount":0.0,"userId":2,"billId":9,"uuidBill":"","uuidUser":null,"delDate":0,"billCurrency":null},{"date":1474934400000,"billName":"xcc","note":null,"startBalance":40.0,"cashAmount":0.0,"userId":2,"billId":10,"uuidBill":"","uuidUser":null,"delDate":0,"billCurrency":null}],"codeOperation":"bills","numSync":null,"lastSyncDate":1468800000,"uuidUser":"4f456aef-f442-336e-9a46-2c13eeae7860","uuiddevice":"fa941af3-e841-3326-b06a-2e9a04cd0e0c"}

// ИЗВЛЕЧЕНИЕ

ObservableList<Bills> billses = (ObservableList<Bills>) mapper.readValue(message, Bills.class);

JsonNode node = jsonParser.getCodec().readTree(jsonParser);

JsonNode bills = node.get("bills");

ArrayNode slaidsNode = (ArrayNode) bills;

Iterator<JsonNode> slaidsIterator = slaidsNode.elements();

while (slaidsIterator.hasNext()) {

Bills bill = new Bills();

JsonNode slaidNode = slaidsIterator.next();

long locDate = slaidNode.get("date").numberValue().longValue();

long delDate = slaidNode.get("deletedDate") != null ? slaidNode.get("deletedDate").numberValue().longValue() : 0;

double stBalance = slaidNode.get("startBalance").numberValue().doubleValue();

double cashAmount = slaidNode.get("cashAmount").numberValue().doubleValue();

int userId = slaidNode.get("userId").numberValue().intValue();

int billId = slaidNode.get("billId").numberValue().intValue();

String note = slaidNode.get("note").asText();

String billName = slaidNode.get("billName").asText();

String uuidBill = slaidNode.get("uuidBill").isNull() ? "" : slaidNode.get("uuidBill").asText();

String uuidUser = slaidNode.get("uuidUser").isNull() ? "" : slaidNode.get("uuidUser").asText();

bill.setDate(locDate);

bill.setBillName(billName);

bill.setNote(note);

bill.setStartBalance(stBalance);

bill.setCashAmount(cashAmount);

bill.setUserId(userId);

bill.setUuidBill(uuidBill);

bill.setUuidUser(uuidUser);

bill.setDelDate(delDate);

bill.setBillId(billId);

billses.add(bill);

}

// ЗАПИСЬ В БАЗУ СЕРВЕРА

CommonDAO.insertNewBills(billses, uuidDevice, responseWithData);

INSERT INTO Bills(UUIDBills, BillName, Date, StartBalance, Note, UUIDUser) VALUES (?,?,?,?,?,?)

# Заключение

В результате выполнения курсового проекта была разработана программа ведения домашней бухгалтерии и управления личными финансами.

В полной мере были решены поставленные задачи:

* исследована предметная область;
* проанализированы существующие решения;
* разработан алгоритм синхронизации.

Программа является кроссплатформенной, многопользовательской и позволяет проводить прозрачную для пользователя синхронизацию через сеть Интернет. Программа предоставляется пользователям бесплатно по лицензии GPL и имеет открытый исходный код.

# Список литературы

1. Egrin U., Building a Sample Java WebSocket Client URL: https://dzone.com/articles/sample-java-web-socket-client (дата обращения: 12.09.2016);
2. Paraschiv E., Getting Started with Custom Deserialization in Jackson URL: http://www.baeldung.com/jackson-deserialization (дата обращения: 9.10.2016);
3. Барчук А., Maven. Знакомство и настройка. URL: http://devcolibri.com/742 (дата обращения: 21.10.2016);
4. Гамма Э., Хелм Р., Джонсон Р., Влиссидес Дж. Приемы объектно-ориентированного программирования. Паттерны проектирования. СПб.: Питер, 2015. 368 с.
5. Голобурдин А., Работа с форматом обмена данными JSON в Java URL:

http://goloburdin.blogspot.ru/2011/03/json-java.html (дата обращения: 8.10.2016);

1. Криль О., Java SQLite ORM URL: http://devcolibri.com/3438 (дата обращения: 13.07.2016);
2. Марко Я., Учебник по JavaFX 8 URL: http://code.makery.ch/library/javafx-8-tutorial/ru/part1/ (дата обращения: 29.08.2016);
3. Семихатов С., Java и формат строк Unicode URL: http://www.javaportal.ru/java/articles/javaandunicode.html (дата обращения: 13.09.2016);
4. Хорстманн Кей С., Корнелл Г. Java. Библиотека профессионала, том 1. Основы. 9-е изд. М.: Издательский Дом «Вильямс», 2014. 864 с.

Минобрнауки России  
федеральное государственное бюджетное образовательное  
учреждение высшего образования  
«Омский государственный университет им. Ф.М. Достоевского»  
Кафедра компьютерных технологий и сетей

УТВЕРЖДАЮ  
Заведующий кафедрой  
\_\_\_\_\_\_\_\_\_\_\_\_\_ Лавров Д. Н.  
«\_\_\_» \_\_\_\_\_\_\_\_\_\_\_ 2016 г.

Разработка приложения для управления личными финансами.  
  
Курсовая работа  
по направлению 230100.62 ­ Информатика и вычислительная техника

Научный руководитель:  
кандидат тех.наук, доцент  
\_\_\_\_\_\_\_\_\_\_ Опарина Т.М.  
«\_\_\_» \_\_\_\_\_\_\_\_\_\_\_ 2016 г.  
Выполнил:  
студент группы СИБ­301У  
\_\_\_\_\_\_\_\_\_ Кузина С.А.  
«\_\_\_» \_\_\_\_\_\_\_\_\_\_\_ 2016 г.

Омск  
2016