МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ

ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Информационных Технологий

Кафедра Информационных систем и технологий

Специальность 1-40 01 01 «Программное обеспечение информационных

технологий»

Специализация Программирование интернет-приложений

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**К КУРСОВОМУ ПРОЕКТУ НА ТЕМУ:**

Разработка базы данных с использованием технологии репликации данных между серверами СУБД для банка

Выполнил студент Муравейко Владислав Сергеевич

(Ф.И.О.)

Руководитель проекта асс. Жигаровская С.А.

(учен. степень, звание, должность, подпись, Ф.И.О.)

Заведующий кафедрой к.т.н., доц. Смелов В.В.

(учен. степень, звание, должность, подпись, Ф.И.О.)

Консультанты асс. Жигаровская С.А.

(учен. степень, звание, должность, подпись, Ф.И.О.)

Нормоконтролер асс. Жигаровская С.А.

(учен. степень, звание, должность, подпись, Ф.И.О.)

Курсовой проект защищен с оценкой

Минск 2018

[Введение 3](#_Toc531844859)

[Постановка задачи 4](#_Toc531844860)

[1. Разработка модели базы данных 5](#_Toc531844861)

[2. Разработка необходимых объектов 7](#_Toc531844862)

[2.1. Таблицы 7](#_Toc531844863)

[2.2. Пользователи 7](#_Toc531844864)

[2.3. Процедуры 7](#_Toc531844865)

[2.4. Функции 8](#_Toc531844866)

[3. Описание процедур импорта и экспорта данных 9](#_Toc531844867)

[3.1. Процедура импорта данных из XML-файла 9](#_Toc531844868)

[3.2. Процедура экспорта данных в XML-файл 9](#_Toc531844869)

[4. Технология репликации Postgresql 10](#_Toc531844870)

[5. Тестирование 11](#_Toc531844871)

[5.1 Тестирование производительности базы данных 11](#_Toc531844872)

[6. Руководство пользователя 13](#_Toc531844873)

[Заключение 14](#_Toc531844874)

[Список использованной литературы 15](#_Toc531844875)

[Приложение А 16](#_Toc531844876)

[Приложение Б 18](#_Toc531844877)

[Приложение В 19](#_Toc531844878)

[Приложение Г 25](#_Toc531844879)

[Приложение Д 26](#_Toc531844880)

# **Введение**

Система управления базами данных (СУБД) – совокупность программных и лингвистических средств общего или специального назначения, обеспечивающих управление созданием и использованием баз данных.

Основные функции СУБД:

* Определение структуры создаваемой базы данных, ее инициализация и проведение начальной загрузки;
* Предоставление пользователям возможности манипулирования данными (выборка необходимых данных, выполнение вычислений, разработка интерфейса ввода/вывода, визуализация);
* Обеспечение логической и физической независимости данных;
* Защита логической целостности базы данных;
* Защита физической целостности;
* Управление полномочиями пользователей на доступ к базе данных;
* Синхронизация работы нескольких пользователей;
* Управление ресурсами среды хранения;
* Поддержка деятельности системного персонала.
* Обычно современная СУБД содержит следующие компоненты:
* Ядро, которое отвечает за управление данными во внешней и оперативной памяти и журнализацию;
* Процессор языка базы данных, обеспечивающий оптимизацию запросов на извлечение и изменение данных и создание, как правило, машинно-независимого исполняемого внутреннего кода;
* Подсистему поддержки времени исполнения, которая интерпретирует программы манипуляции данными, создающие пользовательский интерфейс с СУБД.

СУБД существует огромное множество: Oracle, MS SQL Server, Microsoft Access, MySql и так далее. В данной работе будет использовано решение PostgreSql.

# **Постановка задачи**

На сегодняшний день существует большое количество баз данных. Одной из самых распространённых баз является Postgresql,.

В качестве примера рассмотрим Бд Postgresql

* Возможность создания пользовательских типов данных;
* Множество диалектов для написания процедур и функций;
* Разные типы индексов;
* Полная поддержка транзакционности;
* Высокая производительность и надёжность;
* Возможность компиляции кода.

В наше время почти каждый человек активно использует базы данных в неявном виде.

Таким образом, базы данных ещё многие годы будет актуальным, т.к. оно позволяет пользователям быстро получать информацию из хранилищ без видимой реализации что имеет немаловажное значение.

При разработке Бд были использованы неявные курсоры предствляемые в виде автоматической конструкции for in . Но это не запрещает использовать курсоры в явном виде в Postgresql. Также есть возможность использования разных диалектов для написания кода на Яп прямо в Бд. Имеется множество типов индексов для более точной постановки поиска. В своём проекте я использовал Brin и B-Tree индексы.

Аутентификация пользователя Бд - здесь, аутентификация не ограничивается идентификацией пароля, но является взаимным, то есть аутентификация с использованием хеша требуется, чтобы исключить возможность любого вида посреднической атаки.

Psql – это встроенное API на основе командной строки для администрирования Бд с возможность полного выполнения всех команд, настройки Бд, создания Бд, выполнение резервного копирования Бд, а также взаимодействия с ролями пользователей.

Репликация – технология обмена изменениями данных в Бд между серверами с предустановленной Бд (В нашем случае Postgresql) и выполняется асинхронно и синхронно.

Целостность - окончательный ключевая концепция является сохранение целостности, то есть, предлагая гарантию того, что сообщение не было изменено кем - либо в своем пути от отправителя к получателю.

Для организации работы с базой данных было решено использовать Postgresql.

# **Разработка модели базы данных**

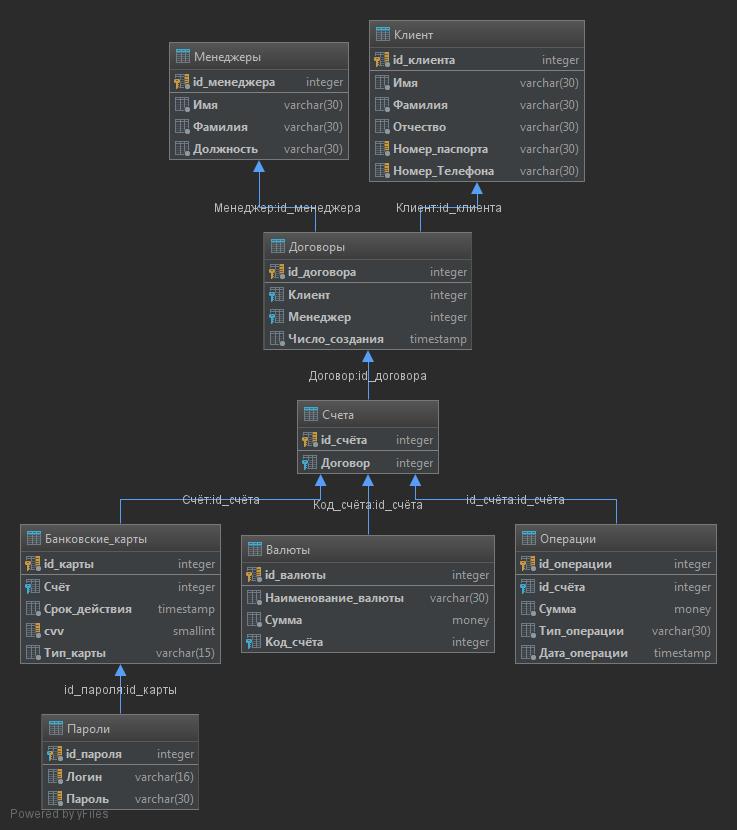


Рисунок 1.1 – Диаграмма базы данных.

Для базы данных банк были разработаны 8 таблиц. Диаграмма связей таблиц для необходимой базы данных представлена н рисунке 1.1.

Таблица Клиент, содержащая клиентов Банка:

* Id\_клиента - уникальный идентификатор;
* Имя­ – имя клиента;
* Фамилия – фамилия клиента;
* Отчество – отчество клиента;
* Номер\_паспорта – номер паспорта клиента;
* Номер\_телефона – номер телефона клиента.

Таблица Менеджеры, менеджеров банка:

* Id\_менеджера – уникальный идентификатор;
* Имя – имя менеджера;
* Фамилия – фамилия менеджера;
* Должность – должность менеджера.

Таблица Договоры, предназначенная для договоров клиента:

* Id\_договора – уникальный идентификатор;
* Клиент – сам клиент;
* Менеджер – менеджер, обслуживающий клиента;
* Число\_создания – дата заполнения договора.

Таблица Счета, предназначенная для хранения счетов клиента:

* Id\_счета – уникальный идентификатор;
* Договор – номер договора, к которому привязан счет;

Таблица Банковские\_карты, предназначенная для хранения банковских карт клиента:

* Id\_карты – уникальный идентификатор;
* Счет – номер счета, к которому привязать карту;
* Срок\_действия – срок действия карты;
* CVV – уникальный номер сзади карты;
* Тип\_карты – тип возможностей карты.

Таблица Пароли, предназначенная для хранения паролей банковской карты:

* Id\_пароля – уникальный идентификатор;
* Логин – логин карты;
* Пароль – пароль карты.

Таблица Валюты, предназначенная для хранения информации о валютах и их сумме на счету:

* Id\_валюты – уникальный идентификатор;
* Наименование\_валюты – наименование валюты;
* Сумма – количество валюты на счету;
* Код\_счета – номер счета для этих валют.

Таблица Операции, предназначена для хранения данных о выполненных операциях со счетом:

* Id\_операции – уникальный идентификатор;
* Id\_счета – номер счета на котором была выполнена операция;
* Сумма – сумма операции;
* Тип\_операции – имя операции;
* Дата\_операции – дата выполнения операции.

Скрипты для создания всех таблиц базы данных представлены в Приложении А.

# **Разработка необходимых объектов**

# **2.1. Таблицы**

Таблицы являются основой любой базы данных, именно в них хранится вся информация. При проектировании базы данных было создано 8 таблиц, которые подробно описаны ранее в разделе 1, а SQL-скрипты для их создания находятся в Приложении А.

# **2.2. Пользователи**

Пользователь базы данных – это физическое или юридическое лицо, которое имеет доступ к БД и пользуется услугами информационной системы для получения информации. На каждом этапе развития БД (проектирование, реализация, эксплуатация, модернизация и развитие, полная реорганизация) с ней связаны разные категории пользователей. При проектировании базы данных понадобилось 2 пользователя (клиент осуществляет операции и администратор просматривающий доп.инфу).

Администратор был наделён только привилегией на выполнение всех хранимых процедур, разработанных для данной базы данных.

Клиенту разрешено выполнять только функции.

Скрипты для создания клиентов базы данных, прочие операции, а также выделение привилегий на осуществление определённых операций с базой данных представлены в приложении Б.

# **Процедуры**

Использование хранимых процедур позволяет ограничить либо вообще исключить непосредственный доступ пользователей к таблицам базы данных, оставив только администраторам на выполнение хранимых процедур, обеспечивающих косвенный и строго регламентированный доступ к данным резервного копирования и сериализации, а также назначению персонала. Листинги некоторых хранимых процедур представлены в приложении В.

Всего было разработано 5 процедуры:

1. Create\_manager – процедура для добавления работников банка;
2. Export\_for\_Database – процедура для экспорта БД в файл \*.xml;
3. Export\_for\_Tables – процедура для экспорта данных в файл \*.xml;
4. Inser – процедура заполнения таблицы Менеджеры данными;
5. Import – процедура для импорта данных в таблицы.

Все скрипты хранимых процедур приложены в отдельных файлах в корне директории прилагаемого диска.

# **Функции**

Использование функций позволяет клиентам выполнять операции над своим счётом, оформлять карту со счетом. Написаны они на диалектах plpgsql и plpythonu. Листинги некоторых хранимых функций представлены в приложении В.

Всего было разработано 10 функций:

1. Add\_money – функция для пополнения счета;
2. Create\_card – функция создания карты клиента;
3. Create\_client – функция создания клиента;
4. Create\_schet – функция создания счета;
5. Is\_exist – функция проверка на принадлежность клиента банку;
6. Poluch\_koof – функция парсинга коэффициента валют с нац. Банка РБ и нац. Банка РФ;
7. Pop\_money – функция для снятие средств со счета;
8. Random\_between – функция для получения рандомного числа от и до;
9. Trade – функция обмена валют;
10. Transact – функция перевода средств со счета на счет.

Все скрипты хранимых функций приложены в отдельных файлах в корне директории прилагаемого диска.

# **Описание процедур импорта и экспорта данных**

# **3.1. Процедура импорта данных из XML-файла**

Для экспорта используется стандартная функция, входящая в набор Postgresql – copy, которую мы модифицировали под наш случай. Пример данного скрипта представлена в Приложении Г.

# **Процедура экспорта данных в XML-файл**

Для работы процедуры экспорта данных в XML-файл мы добавили модификацию экспорта БД обычной функции copy, представленной на рисунке 3.1.

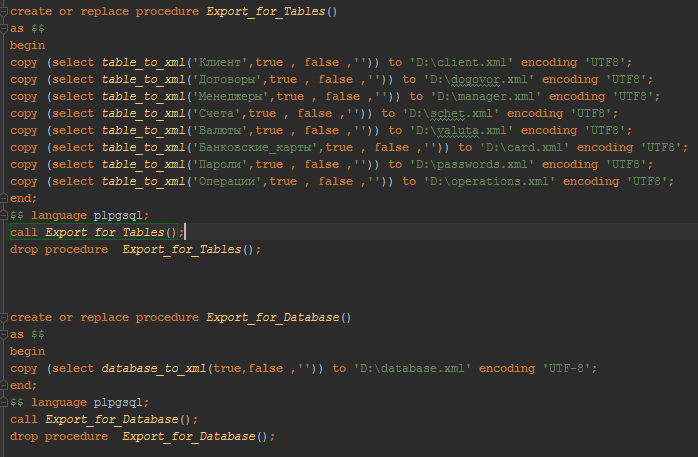


Рисунок 3.1 – Скрипт экспорта данных из таблиц и БД.

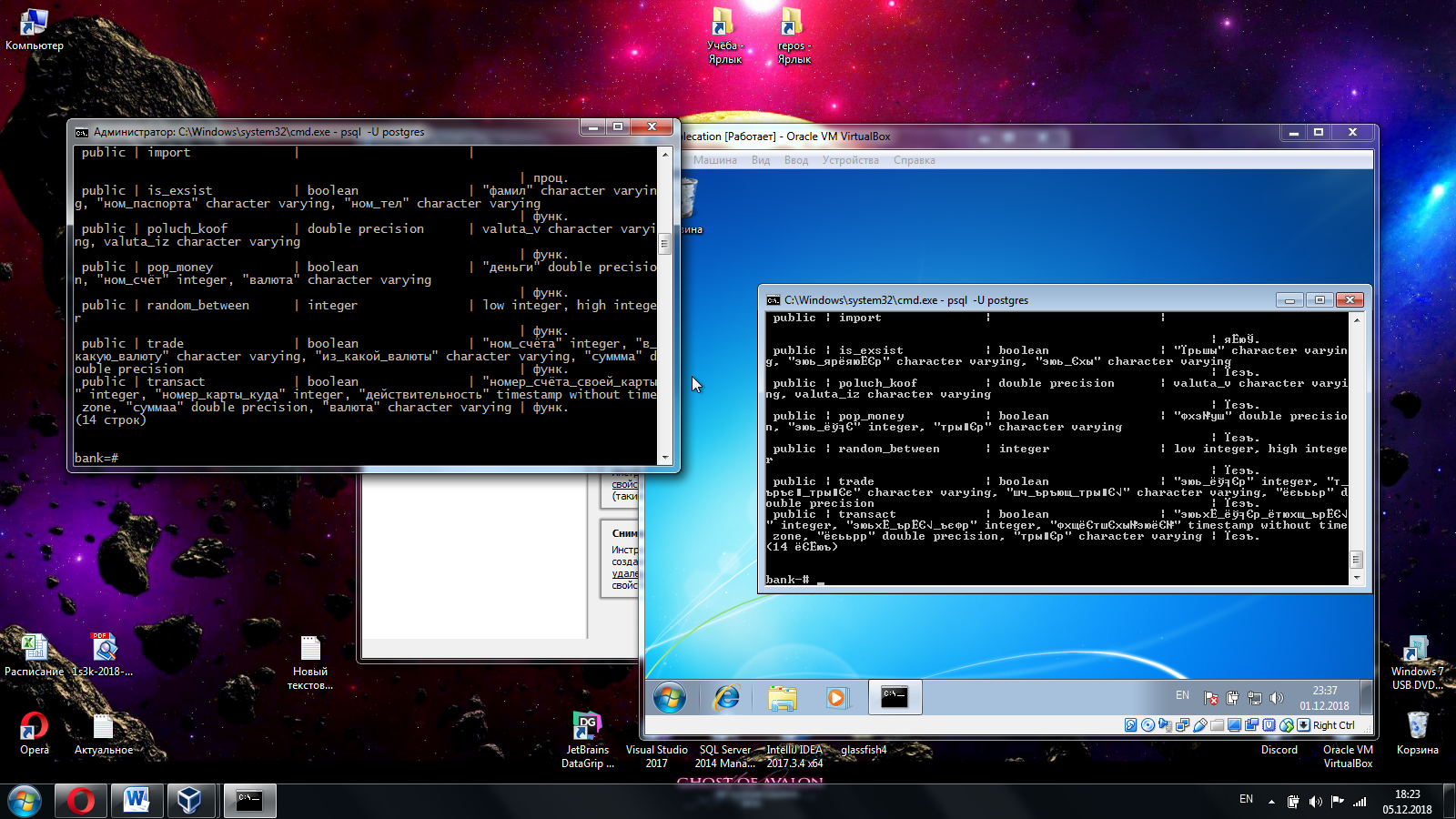
Пример скрипта процедуры экспорта в XML-файл представлен в Приложении Д.

# **Технология репликации Postgresql**

Репликация ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) replication) - механизм [синхронизации](https://ru.wikipedia.org/wiki/%D0%A1%D0%B8%D0%BD%D1%85%D1%80%D0%BE%D0%BD%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D1%8F_(%D0%B8%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0)) содержимого нескольких копий объекта (например, содержимого [базы данных](https://ru.wikipedia.org/wiki/%D0%91%D0%B0%D0%B7%D0%B0_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85)). Репликация — это процесс, под которым понимается копирование данных из одного источника на другой (или на множество других) и наоборот.

При репликации изменения, сделанные в одной копии объекта, могут быть распространены в другие копии. Репликация может быть синхронной или асинхронной. В нашем случае репликация выполняется асинхронно.

Созданный отчет представлен на рисунке 4.1.



4.1 - Отображение функций синхронизируемых между серверами.

Аналогично и для всех остальных возможностей БД, например, изменения в таблицах, добавление процедур, изменения какого-либо объекта БД. Все изменения сохраняются в режиме реального времени.

# **Тестирование**

# **5.1 Тестирование производительности базы данных**

Для тестирования производительности была взята за основу таблица Менеджеры, так как она заполнена гораздо больше всех остальных таблиц.

Изначально таблица Менеджеры была заполнена на 100000 строк инфо. на рисунке 5.1.

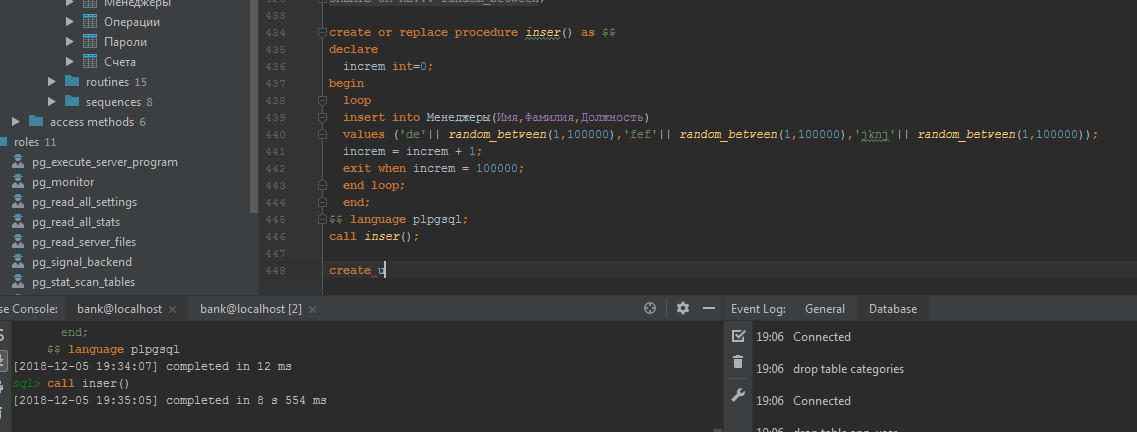


Рисунок 5.1 – Оценка времени заполнения таблицы Менеджеры.

После этого был применён SELECT-запрос к данной таблице и при помощи стандартных средств IDE DataGrip оценена цена выборки к таблице. Результат данной оценки запроса приведён на рисунке 5.2.

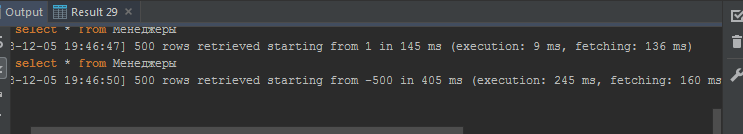


Рисунок 5.2 – Оценка времени селекта таблицы Менеджеры.

После проведения первоначальной оценки был построен B-Tree индекс к таблице Менеджер по столбцу id и проведена оценка такого же SELECT-запроса к таблице Менеджеры. B-tree *(Balanced Tree)* строит индексы используя реализацию сбалансированного дерева. Результаты, полученные во время оценки, представлены на рисунке 5.3.

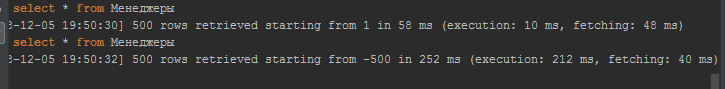


Рисунок 5.3 – Оценка запроса к таблице с построенным B-Tree индексом.

После проведения вторичной оценки был протестирован Brin индекс к таблице Менеджер по столбцу id и проведена оценка такого же SELECT-запроса к таблице Менеджеры. BRIN *(Block Range Index)* предназначается для обработки очень больших таблиц, в которых определённые столбцы некоторым естественным образом коррелируют с их физическим расположением в таблице. Результаты, полученные во время оценки, представлены на рисунке 5.4.

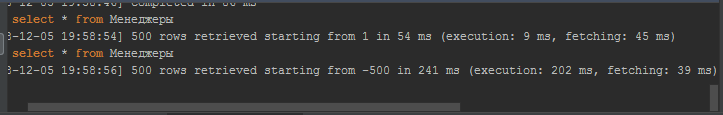


Рисунок 5.4 – Оценка запроса к таблице с построенным Brin индексом.

По результатам проведённых оценок до и после построения B-Tree индекса и после тестирования Brin индекса, можно сделать вывод, что после создания индекса получили небольшой прирост скорости выборки из таблицы при большом наборе данных при помощи построенного сбалансированного дерева, но более выгодным оказался Brin индекс в соответствии с его назначением. Также была ускорена выборка с использование Order by и применение операторов сравнения.

Таким образом, постройка индекса к таблице была более-менее оправдана, так как мы получили прирост производительности и более выгодное распределение ресурсов, в зависимости от параметра, характеризующего запрос.

# **Руководство пользователя**

Для взаимодействия пользователя с Бд были разработаны функции. Пользователи в Бд именуются как Клиенты, тоесть, взаимодействие происходит с клиентами. Вошедший в Банк клиент может зарегистрироваться, чтобы пользоваться услугами банка. При этом, банк предоставляет услуги по регистрации банковских карты, регистрации счетов, заполнения договоров во в купе с регистрацией клиентов, обмена валют, транзакций средств со счёта на счёт и просмотра произведённых операций.

Взаимодействие происходит благодаря функциям. А сам вызов происходит как показано ниже:

select \* from is\_exsist('m','ab','6');

drop function is\_exsist;

select \* from create\_client('v','m','s','abd','6');

drop procedure create\_client(имяя varchar, фам varchar, отчеств varchar, ном\_паспорта varchar, номер\_тел smallint);

call create\_manager('132','efw2ef','12fe');

drop procedure create\_manager(имяя varchar, фамилияя varchar, должност varchar);

select \* from add\_money(1,10.0,'BYN');

drop function add\_money(счёт integer, деньги float, валюта varchar);

select \* from pop\_money(0.03,1,'USD');

drop function pop\_money(деньги float, ном\_счёт integer, валюта varchar);

select \* from transact(1,2,'2021-12-04 18:27:45.456525',1.0,'USD');

drop function transact(номер\_счёта\_своей\_карты integer, номер\_карты\_куда integer,

действительность timestamp, суммаа float, валюта varchar);

select \* from trade(1,'BYN','USD',1.0);

drop function trade(ном\_счёта integer, в\_какую\_валюту varchar, из\_какой\_валюты varchar, суммма float);

select \* from Клиент;

select \* from Менеджеры;

select \* from Договоры;

select \* from Валюты;

select \* from Операции;

select \* from Счета;

select \* from Банковские\_карты;

Выше представлена выборка из таблиц для представления информации для администратора Бд, чтобы он могу соотнести изменения внесённые с изменениями до внесения. Обязательное указание всех полей при внесении информации клиента, иначе будет отказано клиенту в выполнении услуги.

Поэтому, как вы наблюдаете выше, данные фукнции обязательны к заполнению параметров для выполнения их.

Вот функции с их параметрами:

create\_manager(имяя varchar, фамилияя varchar, должност varchar)

procedure create\_client(имяя varchar, фам varchar, отчеств varchar, ном\_паспорта varchar, номер\_тел smallint)

add\_money(счёт integer, деньги float, валюта varchar)

pop\_money(деньги float, ном\_счёт integer, валюта varchar)

transact(номер\_счёта\_своей\_карты integer, номер\_карты\_куда integer,

действительность timestamp, суммаа float, валюта varchar)

trade(ном\_счёта integer, в\_какую\_валюту varchar, из\_какой\_валюты varchar, суммма float)

В итоге, функционал пользователя оказался достаточно прост и информативен для использования.

# **Заключение**

В данном курсовом проекте была разработана база данных для банка. Помимо этого, добавлена возможность выполнить резервное копирование БД

в .xml на случай сбоя работы, а также разобрана технология репликации БД.

В соответствии с полученным результатом, можно сказать, что разработанная программа работает верно, а требования технического задания выполнены в полном объеме.

# **Список использованной литературы**

1. Блинова Е.А. Курс лекций по базам данных / Е.А. Блинова
2. Пацей, Н.В. Технология разработки программного обеспечения / Н.В. Пацей. – Минск: БГТУ, 2016. – 129 с.
3. ProfessorWeb .NET & Web Programming [Электронный ресурс] – Режим доступа: [https://professorweb.ru](https://vk.com/away.php?to=https%3A%2F%2Fprofessorweb.ru&cc_key=) – Дата доступа: 30.11.2018.
4. Оф. Документация к Postgresql [Электронный ресурс] – Режим доступа: <https://postgrespro.ru/docs/postgresql/11/index> - Дата доступа: 01.12.2018.

# **Приложение А**

create Table Клиент(

id\_клиента SERIAL PRIMARY KEY ,

Имя varchar(30) not NULL,

Фамилия varchar(30) not NULL,

Отчество varchar(30) not NULL,

Номер\_паспорта varchar(30) not NULL UNIQUE,

Номер\_Телефона varchar(30) not NULL UNIQUE

);

create Table Договоры(

id\_договора serial PRIMARY KEY,

Клиент integer references Клиент(id\_клиента)

ON DELETE CASCADE ,

Менеджер integer references Менеджеры(id\_менеджера)

On DELETE CASCADE ,

Число\_создания timestamp not null

);

create Table Менеджеры(

id\_менеджера SERIAL PRIMARY KEY ,

Имя varchar(30) not NULL,

Фамилия varchar(30) not NULL,

Должность varchar(30) not NULL

);

create TABLE Счета(

id\_счёта SERIAL PRIMARY KEY,

Договор integer references Договоры(id\_договора)

ON DELETE CASCADE

);

create table Валюты(

id\_валюты SERIAL PRIMARY KEY ,

Наименование\_валюты varchar(30) not NULL check

(Наименование\_валюты ='RUB' or Наименование\_валюты = 'BYN' or

Наименование\_валюты = 'USD'),

Сумма money NOT NULL default 0.0,

Код\_счёта integer references Счета(id\_счёта)

ON DELETE CASCADE

);

create table Пароли(

id\_пароля SERIAL PRIMARY KEY ,

Логин varchar(16) NOT NULL UNIQUE,

Пароль varchar(30) NOT NULL UNIQUE,

FOREIGN KEY (id\_пароля) references

Банковские\_карты(id\_карты) ON DELETE CASCADE

);

create table Операции(

id\_операции SERIAL PRIMARY KEY ,

id\_cчёта integer references Счета(id\_счёта)

ON DELETE CASCADE ,

Сумма money NOT NULL default 0,

Тип\_операции varchar(30) NOT NULL ,

Дата\_операции timestamp not null

);

create table Банковские\_карты(

id\_карты SERIAL PRIMARY KEY ,

Счёт integer references Счета(id\_счёта)

ON DELETE CASCADE ,

Срок\_действия timestamp NOT NULL,

CVV smallint NOT NULL UNIQUE,

Тип\_карты varchar(15) not null check (Тип\_карты = 'Бесконтактная' or

Тип\_карты = 'Контактная') default 'Контактная'

# **Приложение Б**

select \* from Валюты;

insert into Валюты(Наименование\_валюты, Код\_счёта) values ('BYN',1),('USD',1),('RUB',1);

drop table Валюты;

drop table Счета;

drop table Банковские\_карты;

drop table Пароли;

drop table Операции;

drop table Договоры;

drop table Менеджеры;

drop table Клиент;

select \* from Клиент;

select \* from Менеджеры ;

select \* from Договоры;

select \* from Валюты;

select \* from Операции;

select \* from Счета;

select \* from Банковские\_карты;

delete from Клиент values where id\_клиента=14;

delete from Договоры values where id\_договора='2';

delete from Клиент values where id\_клиента='6';

alter table Банковские\_карты alter column Тип\_карты type varchar(15);

create user vlad password 'vlad';

create index For\_Manager on Менеджеры using btree(id\_менеджера);

drop index For\_Manager;

create index For\_Manager on Менеджеры using brin(id\_менеджера);

# **Приложение В**

create or replace procedure Export\_for\_Tables()

as $$

begin

copy (select table\_to\_xml('Клиент',true , false ,'')) to 'D:\client.xml' encoding 'UTF8';

copy (select table\_to\_xml('Договоры',true , false ,'')) to 'D:\dogovor.xml' encoding 'UTF8';

copy (select table\_to\_xml('Менеджеры',true , false ,'')) to 'D:\manager.xml' encoding 'UTF8';

copy (select table\_to\_xml('Счета',true , false ,'')) to 'D:\schet.xml' encoding 'UTF8';

copy (select table\_to\_xml('Валюты',true , false ,'')) to 'D:\valuta.xml' encoding 'UTF8';

copy (select table\_to\_xml('Банковские\_карты',true , false ,'')) to 'D:\card.xml' encoding 'UTF8';

copy (select table\_to\_xml('Пароли',true , false ,'')) to 'D:\passwords.xml' encoding 'UTF8';

copy (select table\_to\_xml('Операции',true , false ,'')) to 'D:\operations.xml' encoding 'UTF8';

end;

$$ language plpgsql;

call Export\_for\_Tables();

drop procedure Export\_for\_Tables();

create or replace procedure Export\_for\_Database()

as $$

begin

copy (select database\_to\_xml(true,false ,'')) to 'D:\database.xml' encoding 'UTF-8';

end;

$$ language plpgsql;

call Export\_for\_Database();

drop procedure Export\_for\_Database();

create or replace procedure Import()

as $$

begin

INSERT INTO Клиент(id\_клиента,Имя,Фамилия,Отчество,Номер\_паспорта,Номер\_Телефона)

SELECT (xpath('//row/id\_клиента/text()', x))[1]::text::int AS id\_клиента,

(xpath('//row/Имя/text()', x))[1]::text::varchar(30) AS Имя,

(xpath('//row/Фамилия/text()', x))[1]::text::varchar(30) AS Фамилия,

(xpath('//row/Отчество/text()', x))[1]::text::varchar(30) AS Отчество,

(xpath('//row/Номер\_паспорта/text()', x))[1]::text::varchar(30) AS Номер\_паспорта,

(xpath('//row/Номер\_Телефона/text()', x))[1]::text::varchar(30) AS Номер\_Телефона

FROM unnest(xpath('//row', pg\_read\_file('D:\client.xml')::xml)) x;

end;

$$ language plpgsql;

select \* from Import('D:\client.xml');

drop function Import();

create or replace procedure inser() as $$

declare

increm int=0;

begin

loop

insert into Менеджеры(Имя,Фамилия,Должность)

values ('de'|| random\_between(1,100000),'fef'|| random\_between(1,100000),'jknj'|| random\_between(1,100000));

increm = increm + 1;

exit when increm = 100000;

end loop;

end;

$$ language plpgsql;

call inser();

create or replace procedure create\_manager(имяя varchar(30),фамилияя varchar(30),должност varchar(30))

as $$

insert into Менеджеры(Имя,Фамилия,Должность)

values(имяя,фамилияя,должност)

$$ language sql;

call create\_manager('132','efw2ef','12fe');

drop procedure create\_manager(имяя varchar, фамилияя varchar, должност varchar);

-----------------------------------------------------Функции---------------------------------------------------------

create or replace function trade(ном\_счёта integer,в\_какую\_валюту varchar(30),

из\_какой\_валюты varchar(30),суммма float) returns boolean

AS $$

declare

koof float = 0;

checkk money = 0;

checkk2 money = 0;

begin

select into koof \* from poluch\_koof(в\_какую\_валюту,из\_какой\_валюты);

select into checkk Сумма from Валюты where Код\_счёта = ном\_счёта

and Наименование\_валюты = из\_какой\_валюты;

select into checkk2 Сумма from Валюты where Код\_счёта = ном\_счёта

and Наименование\_валюты = в\_какую\_валюту;

if checkk = 0.0::numeric::money then

return false;

end if;

if checkk - суммма::numeric::money < 0.0::numeric::money then

return false;

end if;

if в\_какую\_валюту = 'USD' and из\_какой\_валюты = 'BYN' then

update Валюты set Сумма = Сумма + (суммма / koof)::numeric::money where Наименование\_валюты = в\_какую\_валюту

and Код\_счёта = ном\_счёта;

update Валюты set Сумма = Сумма - суммма::numeric::money where Наименование\_валюты = из\_какой\_валюты

and Код\_счёта = ном\_счёта;

if FOUND then

insert into Операции(id\_cчёта,Сумма,Тип\_операции,Дата\_операции) values

(ном\_счёта,(суммма / koof)::numeric::money,'Обмен средств',current\_timestamp);

end if;

return True;

end if;

update Валюты set Сумма = Сумма + (суммма \* koof)::numeric::money where Наименование\_валюты = в\_какую\_валюту

and Код\_счёта = ном\_счёта;

update Валюты set Сумма = Сумма - суммма::numeric::money where Наименование\_валюты = из\_какой\_валюты

and Код\_счёта = ном\_счёта;

if FOUND then

insert into Операции(id\_cчёта,Сумма,Тип\_операции,Дата\_операции) values

(ном\_счёта,(суммма / koof)::numeric::money,'Обмен средств',current\_timestamp);

end if;

return true;

end;

$$ language plpgsql;

create or replace function poluch\_koof(valuta\_v varchar(30),valuta\_iz varchar(30))

returns float

as $$

import urllib

import json

d=False

if valuta\_iz == 'BYN':

if valuta\_v == 'BYN':

return 0.0

response = urllib.urlopen('http://www.nbrb.by/API/ExRates/Rates?Periodicity=0')

data = json.loads(response.read())

for i in data:

for a in i:

if i.get(a)==valuta\_v and a=='Cur\_Abbreviation':

d=True

if d==True and a=='Cur\_OfficialRate':

koof=i.get(a)

return koof

if valuta\_iz == 'RUB':

if valuta\_v == 'RUB':

return 0.0

response = urllib.urlopen('https://www.cbr-xml-daily.ru/daily\_json.js')

data = json.loads(response.read())

for i in data:

if i=='Valute':

b = data.get(i)

for bb in b:

if bb == valuta\_v:

cc = b.get(bb)

for val in cc:

if val == 'Value':

koof=cc.get(val)

return koof

if valuta\_iz == 'USD':

if valuta\_v == 'USD':

return 0.0

if valuta\_v == 'RUB':

response = urllib.urlopen('https://www.cbr-xml-daily.ru/daily\_json.js')

data = json.loads(response.read())

for i in data:

if i=='Valute':

b = data.get(i)

for bb in b:

if bb == 'USD':

cc = b.get(bb)

for val in cc:

if val == 'Value':

koof=cc.get(val)

return koof

if valuta\_v == 'BYN':

response = urllib.urlopen('http://www.nbrb.by/API/ExRates/Rates?Periodicity=0')

data = json.loads(response.read())

for i in data:

for a in i:

if i.get(a)=='USD' and a=='Cur\_Abbreviation':

d=True

if d==True and a=='Cur\_OfficialRate':

koof=i.get(a)

return koof

return 0.0

$$ language plpythonu;

create or replace function transact(номер\_счёта\_своей\_карты integer,номер\_карты\_куда integer,

действительность timestamp,суммаа float,валюта varchar(30))

returns boolean as $$

declare

checkk RECORD;

your money;

begin

select into your Сумма from Валюты where Код\_счёта = номер\_счёта\_своей\_карты

and Наименование\_валюты = валюта;

if your - суммаа::numeric::money < 0::numeric::money then

return false;

else

for checkk in

select id\_карты,Счёт,Срок\_действия from Банковские\_карты

loop

if checkk.id\_карты = номер\_карты\_куда and checkk.Срок\_действия = действительность and суммаа > 0 then

Update Валюты set Сумма = Сумма + суммаа::numeric::money where Код\_счёта = checkk.Счёт

and Наименование\_валюты = валюта;

UPDATE Валюты set Сумма = Сумма - суммаа::numeric::money where Код\_счёта = номер\_счёта\_своей\_карты

and Наименование\_валюты = валюта;

if FOUND then

insert into Операции(id\_cчёта,Сумма,Тип\_операции,Дата\_операции) values

(номер\_счёта\_своей\_карты,суммаа::numeric::money,'Перевод средств',current\_timestamp);

end if;

return true;

end if;

end loop;

end if;

return false;

end;

$$ language plpgsql;

create or replace function add\_money(счёт integer,деньги float,валюта varchar(30))

returns boolean as $$

begin

update Валюты set Сумма = Сумма + деньги::numeric::money where Код\_счёта = счёт

and Наименование\_валюты = валюта;

if FOUND then

insert into Операции(id\_cчёта,Сумма,Тип\_операции,Дата\_операции) values

(счёт,деньги::numeric::money,'Пополнение счёта',current\_timestamp);

end if;

return FOUND;

end;

$$ language plpgsql;

create or replace function pop\_money(деньги float,ном\_счёт integer,валюта varchar(30))

returns boolean as $$

declare tempp money;

begin

select into tempp Сумма from Валюты where Код\_счёта = ном\_счёт

and Наименование\_валюты = валюта;

if tempp - деньги::numeric::money < 0.0::numeric::money then

return FALSE;

else

update Валюты set Сумма = Сумма - деньги::numeric::money where Код\_счёта = ном\_счёт

and Наименование\_валюты = валюта;

if FOUND then

insert into Операции(id\_cчёта,Сумма,Тип\_операции,Дата\_операции) values

(ном\_счёт,деньги::numeric::money,'Снятие средств',current\_timestamp);

end if;

END IF ;

return FOUND;

end;

$$ language plpgsql;

create or replace function create\_card(тип\_карт varchar(15),номер\_счёта integer,

ллогин varchar(30),ппароль varchar(15)) returns integer as $$

begin

insert into Банковские\_карты(Счёт,Срок\_действия,CVV,Тип\_карты)

values(номер\_счёта,current\_timestamp + (3 || 'year')::interval ,random\_between(100,999),тип\_карт);

insert into Пароли(Логин, Пароль)

values (ллогин,ппароль);

return номер\_счёта;

end;

$$ language plpgsql;

create or replace function create\_schet(ид integer)

returns integer

as $$

declare

tempp integer = 0;

begin

insert into Счета(Договор)

values (ид) returning id\_счёта into tempp;

insert into Валюты(Наименование\_валюты, Код\_счёта)

values ('RUB',tempp),('BYN',tempp),('USD',tempp);

return tempp;

end;

$$ language plpgsql;

create or replace function create\_client(имяя varchar(30),фам varchar(30),

отчеств varchar(30),ном\_паспорта varchar(15),

номер\_тел smallint) returns integer as $$

declare

tempp integer;

tempp2 integer;

rww RECORD;

incrementt int = 0;

firstt int = 0;

begin

for rww in select \* from Менеджеры

loop

firstt = min(rww.id\_менеджера);

incrementt = incrementt + 1;

end loop;

insert into Клиент(Имя,Фамилия,Отчество,Номер\_паспорта,Номер\_Телефона)

values(имяя,фам,отчеств,ном\_паспорта,номер\_тел)

returning id\_клиента into tempp;

insert into Договоры(Клиент, Менеджер, Число\_создания)

values (tempp,random\_between(firstt,incrementt),current\_timestamp)

returning id\_договора into tempp2;

return tempp2;

end;

$$ language plpgsql;

CREATE OR REPLACE FUNCTION random\_between(low INT ,high INT)

RETURNS INT

AS $$

BEGIN

RETURN floor(random()\* (high-low + 1) + low);

END;

$$ language plpgsql STRICT;

# **Приложение Г**

create or replace procedure Import()

as $$

begin

INSERT INTO Клиент(id\_клиента,Имя,Фамилия,Отчество,Номер\_паспорта,Номер\_Телефона)

SELECT (xpath('//row/id\_клиента/text()', x))[1]::text::int AS id\_клиента,

(xpath('//row/Имя/text()', x))[1]::text::varchar(30) AS Имя,

(xpath('//row/Фамилия/text()', x))[1]::text::varchar(30) AS Фамилия,

(xpath('//row/Отчество/text()', x))[1]::text::varchar(30) AS Отчество,

(xpath('//row/Номер\_паспорта/text()', x))[1]::text::varchar(30) AS Номер\_паспорта,

(xpath('//row/Номер\_Телефона/text()', x))[1]::text::varchar(30) AS Номер\_Телефона

FROM unnest(xpath('//row', pg\_read\_file('D:\client.xml')::xml)) x;

end;

$$ language plpgsql;

select \* from Import('D:\client.xml');

drop function Import();

create or replace procedure inser() as $$

declare

increm int=0;

begin

loop

insert into Менеджеры(Имя,Фамилия,Должность)

values ('de'|| random\_between(1,100000),'fef'|| random\_between(1,100000),'jknj'|| random\_between(1,100000));

increm = increm + 1;

exit when increm = 100000;

end loop;

end;

$$ language plpgsql;

call inser();

# **Приложение Д**

create or replace procedure Export\_for\_Tables()

as $$

begin

copy (select table\_to\_xml('Клиент',true , false ,'')) to 'D:\client.xml' encoding 'UTF8';

copy (select table\_to\_xml('Договоры',true , false ,'')) to 'D:\dogovor.xml' encoding 'UTF8';

copy (select table\_to\_xml('Менеджеры',true , false ,'')) to 'D:\manager.xml' encoding 'UTF8';

copy (select table\_to\_xml('Счета',true , false ,'')) to 'D:\schet.xml' encoding 'UTF8';

copy (select table\_to\_xml('Валюты',true , false ,'')) to 'D:\valuta.xml' encoding 'UTF8';

copy (select table\_to\_xml('Банковские\_карты',true , false ,'')) to 'D:\card.xml' encoding 'UTF8';

copy (select table\_to\_xml('Пароли',true , false ,'')) to 'D:\passwords.xml' encoding 'UTF8';

copy (select table\_to\_xml('Операции',true , false ,'')) to 'D:\operations.xml' encoding 'UTF8';

end;

$$ language plpgsql;

call Export\_for\_Tables();

drop procedure Export\_for\_Tables();

create or replace procedure Export\_for\_Database()

as $$

begin

copy (select database\_to\_xml(true,false ,'')) to 'D:\database.xml' encoding 'UTF-8';

end;

$$ language plpgsql;

call Export\_for\_Database();

drop procedure Export\_for\_Database();