МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

ФАКУЛЬТЕТ ПРОГРАММНОЙ ИНЖЕНЕРИИ И КОМПЬЮТЕРНОЙ ТЕХНИКИ

КАФЕДРА АППАРАТНО-ПРОГРАММНЫХ КОМПЛЕКСОВ ВЫЧИСЛИТЕЛЬНОЙ ТЕХНИКИ

Курсовая работа

по дисциплине

«Базы данных»

на тему

«Разработка базы данных "Банк"»

Выполнил: студент группы P3455

Федюкович С. А.,

Проверил: Сентерев Ю. А.

Санкт-Петербург

2021г.

Содержание

Задание

Разработать базу данных, которая должна содержать следующие обязательные поля: Номер счёта клиента, Дата открытия счёта, ФИО клиента, Адрес клиента, Сумма на счету, Годовая ставка, Тип операции (приход, расход), Дата операции, ФИО оператора, Должность оператора.

Создать запрос, с помощью которого можно регистрировать приход и расход денежных средств со счёта клиента, с определением общей суммы с учетом процентов (вычисляемые поля). С помощью запросов: составить список операций для заданного клиента (ввод фамилии клиента) за определенный период; найти общее число клиентов, с которыми работал заданный оператор (ввод ФИО оператора) за заданный период; найти клиентов с минимальной и максимальной суммой на счетах.

Создать формы: для ввода информации о клиентах; для вывода информации о движении денежных средств клиента в разные периоды; для вывода информации об операторах и выполненных ими операциях.

Создать отчёт на основании любой таблицы и любого запроса.

Введение

В настоящее время информационные технологии играют далеко не последнюю роль практически во всех сферах деятельности человека. Финансы не исключение. Они буквально построены на информации. Платежи происходят на основе информации, клиент принимает решение о покупке благодаря предоставлению ему необходимой информации. Таким образом, просто необходимо уметь работать с информацией, уметь собирать ее и систематизировать. Компьютеризация банков необходима, чтобы упростить и упорядочить данную систему действий.

Разработка БД, как правило, выполняется для определённой предметной области. Чтобы учесть все ее особенности, проводится изучение предметной области и разрабатывается её формализованное описание. Затем проводится концептуальное моделирования моделирование. Исходя из полученных результатов производится логическое, а после физическое проектирование. Следующим этапом Затем производится разработка и генерация форм, запросов и отчётов.

Предметом данной курсовой работы является разработка базы данных "Банк". Целью курсового проекта является разработка базы данных для автоматизации работы с данными с применением СУБД PostgreSQL 13.2.

1 Проектирование базы данных

Сперва следует определиться с основными сущностями базы данных и чем они характеризуются:

1. Клиент — ФИО и Адрес;
2. Счёт — Номер, Дата открытия, Остаток и Годовая ставка;
3. Оператор — ФИО и должность;
4. Операция — Сумма (положительна или отрицательна в зависимости от прихода или расхода соответственно) и Дата.

Итак, определившись с сущностями, начнём проектирование нашей базы данных с построения концептуальной модели.

1.1 Концептуальная модель БД

Концептуальная модель базы данных — это некая наглядная диаграмма, нарисованная в принятых обозначениях и подробно показывающая связь между объектами и их характеристиками. Создаётся концептуальная модель для дальнейшего проектирования базы данных и перевод её, например, в реляционную базу данных. На концептуальной модели в визуально удобном виде прописываются связи между объектами данных и их характеристиками.

Все объекты, обозначающие вещи, обозначаются в виде прямоугольника. Атрибуты, характеризующие объект — в виде овала, а связи между объектами — ромбами. Мощность связи обозначаются стрелками (в направлении, где мощность равна многим — двойная стрелка, а со стороны, где она равна единице — одинарная).

Концептуальная модель, выполненная по заданию, изображена на Рисунке 1.

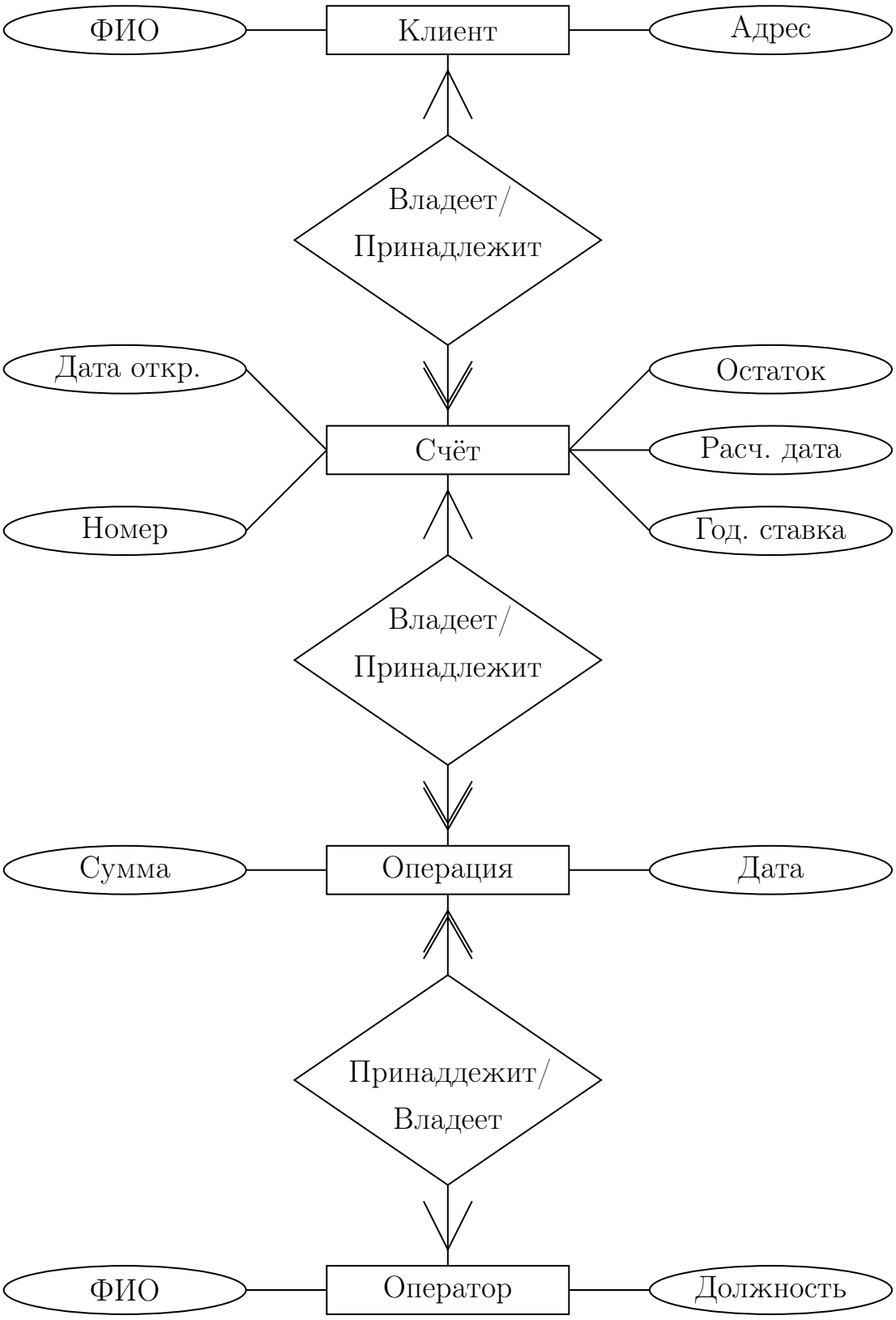

Рисунок 1: Схема концептуальной модели базы данных "Банк"

Следующим этапом после построения концептуальной модели является логическое проектирование.

1.2 Логическая модель БД

На этом этапе мы получаем модель данных.

Логическая модель — графическое представление структуры базы данных с учетом принимаемой модели данных (иерархической, сетевой, реляционной и т. д.), независимое от конечной реализации базы данных и аппаратной платформы. Иными словами, она показывает, ЧТО хранится в базе данных (объекты предметной области, их атрибуты и связи между ними), но не отвечает на вопрос КАК.

Для любой предметной области существует множество вариантов проектных решений её отображения в логической модели. Методика проектирования должна обеспечивать выбор наиболее подходящего проектного решения.

Этап создания логической модели называется логическим проектированием. При переходе от концептуальной (инфологической) модели к логической (даталогической) следует иметь в виду, что концептуальная модель должна включать в себя всю информацию о предметной области, необходимую для проектирования базы данных. Преобразование концептуальной модели в логическую, осуществляется по формальным правилам. Логическая модель базы данных строится в терминах информационных единиц, допустимых в той конкретной СУБД, в среде которой мы проектируем базу данных.

Описание логической структуры базы данных на языке СУБД называется схемой данных.

Спроектировать логическую структуру базы данных означает определить все информационные единицы и связи между ними, задать их имена; если для информационных единиц возможно использование разных типов, то определить их тип. Следует также задать некоторые количественные характеристики, например длину поля.

После определения структуры системы (БД): объектов (таблиц), состава

их полей (структуры таблиц) и связей между таблицами приступают к непосредственному формированию структуры таблиц и определяют ключевые поля в них.

Связь «один-к-многим» в реляционной модели реализуется с помощью полей внешнего ключа. В нашем случае это таблицы:

1. «Клиенты-Счета» — в ней колонка таблицы «Счета» ClientId ссылается на колонку id таблицы «Клиенты» с помощью внешнего ключа.

2. «Счета-Операции» — в ней колонка таблицы «Операции» AccountId ссылается на колонку id таблицы «Счета» с помощью внешнего ключа.

3. «Операторы-Операции» — в ней колонка таблицы «Операции» OperatorId ссылается на колонку id таблицы «Операторы» с помощью внешнего ключа.

Приведём характеристики полей таблиц всех сущностей.

Таблица «Клиенты»		
Имя поля	Тип данных	Размер поля
Id	Числовой	Целые
ФИО	Текстовый	50
Адрес	Текстовый	100

Таблица 1: Характеристика полей таблицы «Клиенты»

Таблица «Счета»		
Имя поля	Тип данных	Размер поля
id	Числовой	Целые
client_id	Числовой	Целые
Дата открытия	Дата	-
Расчётная дата	Дата	-
Номер	Текстовый	10
Остаток	Числовой	Дробные
Годовая ставка	Числовой	Дробные

Таблица 2: Характеристика полей таблицы «Счета»

Таблица «Операции»		
Имя поля	Тип данных	Размер поля
id	Числовой	Целые
account_id	Числовой	Целые
operator_id	Числовой	Целые
Сумма	Числовой	Дробные
Дата	Дата	-

Таблица 3: Характеристика полей таблицы «Операции»

Таблица «Операторы»		
Имя поля	Тип данных	Размер поля
Id	Числовой	Целые
ФИО	Текстовый	50
Должность	Дата	30

Таблица 4: Характеристика полей таблицы «Операторы»

1.3 Физическая модель

Для привязки логической модели к среде хранения используется модель данных физического уровня, для краткости часто называемая физической моделью. Эта модель определяет способы физической организации данных в среде хранения. Модель физического уровня строится с учётом возможностей, предоставляемых СУБД. Описание физической структуры базы данных называется схемой хранения. Соответствующий этап проектирования БД называется физическим проектированием.

Физическая модель данных описывает данные средствами конкретной СУБД. На этапе физического проектирования учитывается специфика конкретной модели данных и специфика конкретной СУБД. Отношения, разработанные на стадии формирования логической модели данных, преобразуются в таблицы, атрибуты становятся столбцами таблиц, для ключевых атрибутов создаются уникальные индексы, домены преображаются в типы данных, принятые в используемой СУБД. Ограничения, имеющиеся в логической модели данных, реализуются различными средствами СУБД, например при помощи индексов, ограничений целостности, триггеров, хранимых процедур. При этом решения, принятые на уровне логического моделирования, определяют некоторые границы, в пределах которых можно развивать физическую модель данных. Точно также в пределах этих границ можно принимать различные решения.

Следующий SQL код создаст данную схему в базе данных:

```
CREATE TABLE Клиенты
(
    id    SERIAL PRIMARY KEY,
    ФИО   VARCHAR(50)  NOT NULL,
    Адрес VARCHAR(100) NOT NULL
);

CREATE TABLE Счета
(
    id                SERIAL PRIMARY KEY,
    client_id         INT            NOT NULL REFERENCES
     ↪ Клиенты ON DELETE CASCADE,
    "Дата открытия"   TIMESTAMP      NOT NULL DEFAULT
     ↪ CURRENT_TIMESTAMP,
    "Расчётная дата" TIMESTAMP       NOT NULL DEFAULT
     ↪ CURRENT_TIMESTAMP,
    Номер             VARCHAR(10)    NOT NULL,
    Остаток           numeric(17, 2) NOT NULL DEFAULT 0,
    "Годовая ставка" numeric(4, 2)   NOT NULL DEFAULT 0
);

CREATE TABLE Операторы
(
    id        SERIAL PRIMARY KEY,
    ФИО       VARCHAR(50)  NOT NULL,
    Должность VARCHAR(30) NOT NULL
);

CREATE TABLE Операции
(
```

```
    id          SERIAL PRIMARY KEY,
    account_id  INT             NOT NULL REFERENCES Счета ON
     ↪  DELETE CASCADE,
    operator_id INT             NULL REFERENCES Операторы ON
     ↪  DELETE CASCADE,
    Сумма       numeric(16, 2) NOT NULL,
    Дата        TIMESTAMP       NOT NULL DEFAULT
     ↪  CURRENT_TIMESTAMP
);
```

2 Нормализация данных

Нормализация представляет собой процесс разделения данных по отдельным связанным таблицам. Нормализация устраняет избыточность данных (data redundancy) и тем самым позволяет избежать нарушения целостности данных при их изменении, то есть избежать аномалий изменения (update anomaly). В ненормализованной форме таблица может хранить информацию о двух и более сущностях. Также она может содержать повторяющиеся столбцы. Также столбцы могут хранить повторяющиеся значения. В нормализованной же форме каждая таблица хранит информацию только об одной сущности.

В нашем случае присутствует избыточность, так как остаток на счету хранится, как сумма операций и как поле счёта, но данная избыточность сделана намеренно в целях оптимизации, потому что каждый раз считать сумму операций будет дорого, особенно с ростом количества операций. Для избежания аномалий (сумма операций не равна остатку), все операции над остатком будут проводиться внутри транзакций.

3 Работа с базой данных с помощью CLI psql

CLI программа psql — это терминальный клиент для работы с PostgreSQL. Она позволяет интерактивно вводить запросы, передавать их в PostgreSQL и видеть результаты. Также запросы могут быть получены из файла или из аргументов командной строки. Кроме того, psql предоставляет ряд метакоманд и различные возможности, подобные тем, что имеются у командных оболочек, для облегчения написания скриптов и автоматизации широкого спектра задач.

3.1 Подключение к базе данных

psql это клиент для PostgreSQL. Для подключения к базе данных нужно указать имя базы данных, имя сервера, номер порта сервера и имя пользователя через аргументы командной строки -d, -h, -p и -U соответственно. Если в командной строке есть аргумент, который не относится к параметрам psql, то он используется в качестве имени базы данных (или имени пользователя, если база данных уже задана). У всех аргументов есть значения по умолчанию. Если опустить имя сервера, psql будет подключаться через Unix-сокет к локальному серверу, либо подключаться к localhost по TCP/IP в системах, не поддерживающих UNIX-сокеты. Номер порта по умолчанию определяется автоматически. Имя пользователя по умолчанию, как и имя базы данных по умолчанию, совпадает с именем пользователя в операционной системе.

Пример подключения к базе данных:

```
$ psql -H 127.0.0.1 -U root course
```

3.2 Ввод SQL-команд

Как правило, приглашение psql состоит из имени базы данных, к которой psql в данный момент подключён, а затем строки =>. Например:

```
$ psql course
psql (9.6.22)
Type "help" for help.

root=>
```

В командной строке пользователь может вводить команды SQL. Обычно введённые строки отправляются на сервер, когда встречается точка с запятой, завершающая команду. Конец строки не завершает команду. Это позволяет разбивать команды на несколько строк для лучшего понимания. Если команда была отправлена и выполнена без ошибок, то результат команды выводится на экран.

Пример команды:

```
root => CREATE TABLE Клиенты(
        Id      SERIAL PRIMARY KEY,
        ФИО     VARCHAR(50) NOT NULL,
        Адрес   VARCHAR(100) NOT NULL
    );
CREATE TABLE
root =>
```

3.3 Определение функций и процедур

Далее будут использоваться функции и процедуры для описания всех операций над сущностями в базе.

Для объявления процедур используется следующий синтаксис:

```
CREATE [OR REPLACE] PROCEDURE name(...arguments)
    LANGUAGE plpgsql
AS
$$
DECLARE
    ...
BEGIN
    ...
END;
$$
```

А для объявление функций, которые отличаются от процедур только наличием возможности вернуть какое-либо значение, будет использоваться следующий синтаксис:

```
CREATE [OR REPLACE] FUNCTION name(...arguments) RETURNS value
    LANGUAGE plpgsql
AS
$$
DECLARE
    ...
BEGIN
    ...
END;
$$
```

3.4 Работа с операциями и счётом

Для регистрации пополнений и списаний со счёта была написана следующая процедура:

```
CREATE OR REPLACE PROCEDURE charge(account_id_to_charge INT,
 ↪  amount numeric, charge_operator_id INT)
    LANGUAGE plpgsql
AS
$$
DECLARE
BEGIN
    INSERT INTO "Операции"(account_id, operator_id, "Сумма",
     ↪  "Дата")
    VALUES (account_id_to_charge, charge_operator_id, amount,
     ↪  CURRENT_TIMESTAMP);
    UPDATE "Счета" SET "Остаток" = "Остаток" + amount WHERE id
     ↪  = account_id_to_charge;
    COMMIT;
END;
$$
```

Данная процедура принимает несколько аргументов: идентификатор счёта, над которым необходимо провести операцию; сумма операции (отрицательна для списаний или положительна для пополнений); идентификатор оператора, который проводит операцию (null если операцию проводит банк);

Для получения остатка по счёту в конкретный момент времени была написана следующая функция:

```
CREATE OR REPLACE FUNCTION get_total_for_period(id_account
 ↪  INT, total_date TIMESTAMP) RETURNS numeric
    LANGUAGE plpgsql
```

```
AS
$$
DECLARE
    result numeric;
BEGIN
    SELECT COALESCE(SUM("Сумма"), 0)
    INTO result
    FROM "Операции"
    WHERE "Дата" <= total_date
      AND account_id = id_account;
    RETURN result;
END;
$$
```

Данная функция принимает два аргумента: идентификатор счёта и момент времени, в который необходимо получить остаток.

Для начисления процентов по остатку была написана следующая процедура:

```
CREATE PROCEDURE charge_interests()
    LANGUAGE plpgsql
AS
$$
DECLARE
    cur_accounts          REFCURSOR;
    cur_operations        REFCURSOR;
    account_to_charge_id  INT;
    months_to_charge      INT;
    period                INT;
    now                   TIMESTAMP;
    charge_day            TIMESTAMP;
    next_charge_day       TIMESTAMP;
```

```
    start_period          TIMESTAMP;
    operation_day         TIMESTAMP;
    interests             numeric;
    interests_per_second numeric;
    operation_sum         numeric;
    total                 numeric;
    percents_for_month    numeric;
    total_percents        numeric;
    percents              numeric;
BEGIN
    now := CURRENT_TIMESTAMP;
    OPEN cur_accounts FOR SELECT id, "Расчётная дата",
     ↪  "Годовая ставка"
                          FROM "Счета"
                          WHERE (DATE_PART('year', AGE(now,
                           ↪  "Расчётная дата")) * 12 +
                              DATE_PART('month', AGE(now,
                               ↪  "Расчётная дата"))) > 1
                             FOR UPDATE;
    LOOP
        FETCH cur_accounts INTO account_to_charge_id,
         ↪  charge_day, interests;
        EXIT WHEN NOT FOUND;

        months_to_charge := DATE_PART('year', AGE(now,
         ↪  charge_day)) * 12 +
                            DATE_PART('month', AGE(now,
                             ↪  charge_day));
        next_charge_day := charge_day + interval '1 month';
        total_percents := 0;
        total := get_total_for_period(account_to_charge_id,
         ↪  charge_day);
```

```
interests_per_second := interests / 3153600000.0;

FOR i IN 1..months_to_charge
    LOOP
        start_period := charge_day;
        percents_for_month := 0;
        OPEN cur_operations FOR SELECT "Сумма", "Дата"
                                FROM "Операции"
                                WHERE account_id =
                                 ↪  account_to_charge_id
                                  AND "Дата" >
                                   ↪  charge_day
                                  AND "Дата" <=
                                   ↪  next_charge_day
                                ORDER BY "Дата";
        LOOP
            FETCH cur_operations INTO operation_sum,
             ↪  operation_day;
            EXIT WHEN NOT FOUND;

            IF percents > 0 THEN
                percents_for_month :=
                 ↪  percents_for_month + percents;
            END IF;
            total := total + operation_sum;
            period := EXTRACT(EPOCH FROM
             ↪  (operation_day - start_period));
            percents := total * interests_per_second *
             ↪  period;
            start_period := operation_day;
        END LOOP;
        CLOSE cur_operations;
```

```
                period := EXTRACT(EPOCH FROM (next_charge_day
                ↪ - start_period));
                percents := total * interests_per_second *
                ↪ period;
                IF percents > 0 THEN
                    percents_for_month := percents_for_month +
                    ↪ percents;
                END IF;

                charge_day := charge_day + interval '1 month';
                next_charge_day := charge_day + interval '1
                ↪ month';
                total_percents := total_percents +
                ↪ percents_for_month;

                INSERT INTO "Операции"(account_id,
                ↪ operator_id, "Сумма", "Дата")
                VALUES (account_to_charge_id, null,
                ↪ percents_for_month, charge_day);
            END LOOP;

        UPDATE "Счета"
        SET "Остаток"        = "Остаток" + total_percents,
            "Расчётная дата" = next_charge_day
        WHERE id = account_to_charge_id;
    END LOOP;

    CLOSE cur_accounts;
END;
$$
```

Данная процедура не имеет аргументов. При запуске, процедура находит счета, которым необходимо начислить проценты, а дальше считает проценты для каждого месяца, учитывая, что остаток в каждый определённый день может быть разным. Такую процедуру необходимо запускать каждую минуту, чтобы остаток начислялся во время. Данная функция оптимизирована, а так же использует построчные блокировки, поэтому частый запуск не вызовет проблем при использовании.

3.5 Получение данных

Для получения информации о движении средств клиента по ФИО, была разработана следующая функция:

```
CREATE OR REPLACE FUNCTION get_client_operations(full_name
 ↪  VARCHAR, start_period TIMESTAMP, end_period TIMESTAMP)
 ↪  RETURNS setof "Операции"
    LANGUAGE plpgsql
AS
$$
DECLARE
    r "Операции"%rowtype;
BEGIN
    FOR r IN SELECT "Операции".id, account_id, operator_id,
     ↪  "Сумма", "Дата"
              FROM "Операции"
                       LEFT JOIN "Счета" C on C.id =
                        ↪  "Операции".account_id
                       LEFT JOIN "Клиенты" K on K.id =
                        ↪  C.client_id
              WHERE "ФИО" = full_name
                AND "Дата" >= start_period
                AND "Дата" <= end_period
        LOOP
            RETURN NEXT r;
        END LOOP;
    RETURN;
END;
$$
```

Данная функция имеет три аргумента: ФИО клиента и два аргумента даты для задания интервала.

Полностью аналогично была написана функция для вывода операций конкретного оператора по ФИО:

```
CREATE OR REPLACE FUNCTION get_operator_operations(full_name
 ↪  VARCHAR, start_period TIMESTAMP, end_period TIMESTAMP)
 ↪  RETURNS setof "Операции"
    LANGUAGE plpgsql
AS
$$
DECLARE
    r "Операции"%rowtype;
BEGIN
    FOR r IN SELECT "Операции".id, account_id, operator_id,
     ↪  "Сумма", "Дата"
              FROM "Операции"
                       LEFT JOIN "Операторы" O on O.id =
                        ↪  "Операции".operator_id
              WHERE "ФИО" = full_name
                AND "Дата" >= start_period
                AND "Дата" <= end_period
        LOOP
            RETURN NEXT r;
        END LOOP;
    RETURN;
END;
$$
```

Для вывода всех клиентов, с которыми работал оператор, была разработана следующая функция с таким же набором аргументов:

```
CREATE OR REPLACE FUNCTION get_clients(full_name VARCHAR,
 ↪  start_period TIMESTAMP, end_period TIMESTAMP) RETURNS
 ↪  setof "Клиенты"
```

```
    LANGUAGE plpgsql
AS
$$
DECLARE
    r "Клиенты"%rowtype;
BEGIN
    FOR r IN SELECT *
             FROM "Клиенты"
             WHERE id IN (SELECT client_id
                          FROM "Операторы"
                                   LEFT JOIN "Операции" O on
                                    ↪  "Операторы".id =
                                    ↪  O.operator_id
                                   LEFT JOIN "Счета" C on C.id
                                    ↪  = O.account_id
                          WHERE "Операторы"."ФИО" = full_name
                            AND "Дата" >= start_period
                            AND "Дата" <= end_period
             )
         LOOP
             RETURN NEXT r;
         END LOOP;
    RETURN;
END;
$$
```

Для получения клиента с минимальным и максимальным остатком на счетах был написан следующий запрос:

```
SELECT (SELECT client_id FROM "Счета" order by "Остаток" LIMIT
 ↪  1),
```

```
        (SELECT client_id FROM "Счета" order by "Остаток" desc
        ↪  LIMIT 1)
```

Для добавления клиентов в базу была написана следующая функция, имеющая два аргумента ФИО и Адреса клиента:

```
CREATE OR REPLACE PROCEDURE create_client(full_name VARCHAR,
 ↪  address VARCHAR)
    LANGUAGE plpgsql
AS
$$
DECLARE
BEGIN
    INSERT INTO "Клиенты"("ФИО", "Адрес") VALUES (full_name,
     ↪  address);
END;
$$
```

3.6 Пример работы БД

PostgreSQL не имеет форм и отчётов, поэтому дальше будет приведён пример работы с разработанной базой с использованием всех таблиц, функций и процедур, описанных ранее.

Сперва создадим несколько клиентов при помощи описанной ранее процедуры:

```
postgres=# call create_client('Клиент 1', 'Адрес 1');
CALL
postgres=# call create_client('Клиент 2', 'Адрес 2');
CALL
postgres=# call create_client('Клиент 3', 'Адрес 3');
CALL
postgres=# call create_client('Клиент 4', 'Адрес 4');
CALL
postgres=# call create_client('Клиент 5', 'Адрес 5');
CALL
```

Далее добавим некоторым клиентам счета:

```
postgres=# INSERT INTO "Счета"
            (client_id, "Номер", "Годовая ставка")
           VALUES
            (1, '0000000001', 10.0),
            (2, '0000000002', 25.0),
            (3, '0000000003', 50.0);
INSERT 0 3
```

И так же создадим несколько операторов:

```
postgres=# INSERT INTO "Операторы"
            ("ФИО", "Должность")
```

```
            VALUES
             ('Оператор 1', 'Должность 1'),
             ('Оператор 2', 'Должность 2');
INSERT 0 2
```

Дальше проведём несколько операций и проанализируем результаты:

```
postgres=# call charge(1, 50, 1);
CALL
postgres=# call charge(1, 100, 1);
CALL
postgres=# call charge(1, -10, 1);
CALL
postgres=# call charge(1, 10, 2);
CALL
postgres=# call charge(2, 10, 2);
CALL
postgres=# SELECT *
           FROM get_clients(
            'Оператор 2',
            '2021-06-19 15:19:20.0',
            '2021-06-19 22:19:20.0'
           );
 id |   ФИО    |  Адрес
----+----------+---------
  1 | Клиент 1 | Адрес 1
  2 | Клиент 2 | Адрес 2
(2 rows)
postgres=# SELECT (
            SELECT client_id
            FROM "Счета"
            ORDER BY "Остаток"
```

```
            LIMIT 1
           ),
           (
            SELECT client_id
            FROM "Счета"
            ORDER BY "Остаток" DESC
            LIMIT 1
           );
 client_id | client_id
-----------+-----------
         3 |         1
(1 row)
postgres=# SELECT id, "Сумма"
           FROM get_client_operations('Клиент 1', '2021-06-19
            ↪  15:19:20.0', '2021-06-19 22:19:20.0');
 id | Сумма
----+--------
  4 |  10.00
  3 | -10.00
  2 | 100.00
  1 |  50.00
(4 rows)
postgres=# SELECT id, "Сумма"
           FROM get_operator_operations('Оператор 1',
            ↪  '2021-06-19 15:19:20.0', '2021-06-19
            ↪  22:19:20.0');
 id | Сумма
----+--------
  1 |  50.00
  2 | 100.00
  3 | -10.00
(3 rows)
```

```
postgres=# call charge_interests();
CALL
```

В результате, все операции сработали правильно.

Заключение

Таким образом в данной работе была продемонстрирована разработка базы данных для конкретной предметной области. Были разобраны следующие аспекты: Определение таблиц, основные операции, а также было показано как можно связывать две сущности разными связями.

Реляционные СУБД всё ещё доминируют на рынке, поэтому велика вероятность того, что на сервере используется Реляционная СУБД, например PostgreSQL. Данная база данных хорошо показывает себя на практике и может использоваться почти в любой предметной области.

В разработке проектов PostgreSQL используется несколько реже, чем MySQL / MariaDB, но всё же эта пара с заметным отрывом опережает по частоте использования остальные системы управления базами данных. При этом в разработке больших приложений PostgreSQL опережает по использованию MySQL и MariaDB. Большинство фреймворков (например, Ruby on Rails, Yii, Symfony, Django) поддерживают использование PostgreSQL в разработке.

Список литературы

1. Архитектура и технологии IBM Server Series / В.А. Варфоломеев и др. - М.: Интернет-университет информационных технологий, 2015. - 640 с.

2. Владимир, Михайлович Илюшечкин Основы использования и проектирования баз данных / Владимир Михайлович Илюшечкин. - М.: Юрайт, 2015. - 516 с.

3. Ноубл, Дж., Андерсон, Т., Брэйтуэйт, Г., Казарио, М., Третола, Р. Flex 4. Рецепты программирования. — БХВ-Петербург, 2011. — С. 548. — 720 с.

4. Кузин, А.В. Базы данных: Учебное пособие для студ. высш. учеб. заведений / А.В. Кузин, С.В. Левонисова. - М.: ИЦ Академия, 2012. - 320 с.

5. Илюшечкин, В. М. Основы использования и проектирования баз данных / В.М. Илюшечкин. - М.: Юрайт, Юрайт, 2013. - 224 с.

6. Зубов, А. В. Основы искусственного интеллекта для лингвистов / А.В. Зубов, И.И. Зубова. - Москва: РГГУ, 2013. - 320 с.

7. Стружкин, Н. П. Базы данных. Проектирование. Учебник / Н.П. Стружкин, В.В. Годин. - М.: Юрайт, 2016. - 478 с.

8. Моргунов, Е.П. PostgreSQL. Основы языка SQL: Учебно-практическое пособие, 2019. 336 с.

9. Голицына, О. Л. Базы данных / О.Л. Голицына, Н.В. Максимов, И.И. Попов. - М.: Форум, 2015. - 400 с.