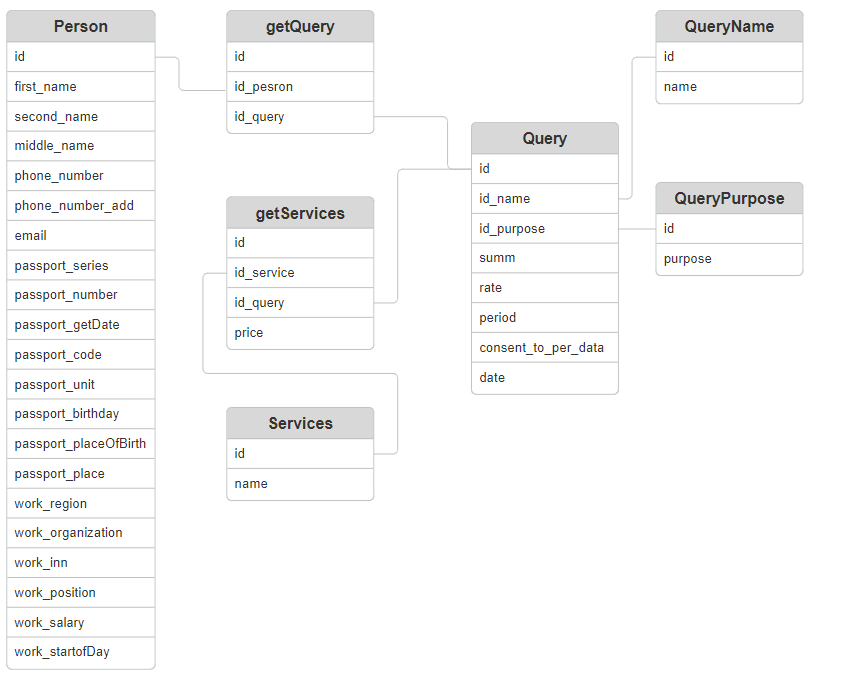
**Решение тестового задания (Аналитик-проектировщик)**

I. Использовались следующие технологии:

1. Реляционная база данных PostgreSQL 16.2
2. Язык программирования JS

**Задание 1**

I. ER-диаграмма:



II. Описание сущностей и атрибутов

1. Для данной модели данных выбраны 7 основных сущностей:

- **Person** – описывает такие данные клиента, как контактные и паспортные данные, сведения о месте работы;

- **Query** – описывает заявки клиента;

- **Services** – описывает дополнительные сервисы, которыми может воспользоваться клиент. Это могут быть, например, страхование жизни или юридическая помощь;

- **getQuery** – отражает связь между клиентом и заявкой;

- **getSevices** – отражает связь между клиентом и дополнительными сервисами;

- **QueryName** – отражает наименование продукта. Количество продуктов ограничено. Для экономии памяти следует выделить их в отдельную таблицу.

- **QueryPurpose** – отражает цель взятия займа. Количество целей ограничено. Для экономии памяти следует выделить их в отдельную таблицу.

2. В следующих таблицах представлены атрибуты сущностей и тип данных.

Таблица 1. Сущность Person

|  |  |  |  |
| --- | --- | --- | --- |
| **Сущность-атрибут** | **Описание** | **Тип данных** | **Причина выбора** |
| Person.id | Идентификатор клиента | integer | Тип данных integer позволяет хранить целые числа с верхним пределом 2147483647, занимает 4 байта. |
| Person.first\_name | Имя заемщика | varchar(100) | Для имени заемщика можно использовать тип данных varchar с ограничением длины в 100 символов. varchar(n) позволяет хранить строки переменной длины, что и требуется, поскольку данные пользователей могут иметь разную длину |
| Person.second\_name | Фамилия заемщика | varchar(100) | Можно использовать тип данных varchar с ограничением длины в 100 символов. varchar(n) позволяет хранить строки переменной длины, что и требуется, поскольку данные пользователей могут иметь разную длину |
| Person.middle\_name | Отчество заемщика | varchar(100) | Можно использовать тип данных varchar с ограничением длины в 100 символов. varchar(n) позволяет хранить строки переменной длины, что и требуется, поскольку данные пользователей могут иметь разную длину |
| Person.phone\_number | Номер мобильного телефона заемщика | varchar(20) | Можно использовать тип данных varchar с ограничением длины в 20 символов. varchar(n) позволяет хранить строки переменной длины, что и требуется, поскольку данные пользователей могут иметь разную длину |
| Person. phone\_number\_add | Дополнительный номер мобильного телефона заемщика | varchar(20) | Можно использовать тип данных varchar с ограничением длины в 20 символов. varchar(n) позволяет хранить строки переменной длины, что и требуется, поскольку данные пользователей могут иметь разную длину |
| Person. email | Электронная почта заемщика | varchar(100) | Можно использовать тип данных varchar с ограничением длины в 100 символов. varchar(n) позволяет хранить строки переменной длины, что и требуется, поскольку данные пользователей могут иметь разную длину |
| Person. passport\_series | Серия паспорта заемщика | char(4) | Серия паспорта – это фиксированное значение, состоящее из 4 цифр. Значение может начинаться с нуля, поэтому integer использовать нельзя, поскольку 0 будут исключены. Здесь целесообразнее использовать тип char с ограничением в 4 символа |
| Person. passport\_number | Номер паспорта заемщика | char(6) | Номер паспорта – это фиксированное значение, состоящее из 6 цифр. Значение может начинаться с нуля, поэтому integer использовать нельзя, поскольку 0 будут исключены. Здесь целесообразнее использовать тип char с ограничением в 6 символов |
| Person. passport\_getDate | Дата выдачи паспорта | date | Позволяет хранить дату в формате YYYY-MM-DD. В отличие от timestamp занимает всего 4 байта |
| Person. passport\_code | Код подразделения | char(7) | Код подразделения – это фиксированное значение, состоящее из пары чисел, разделенных тире. Здесь целесообразнее использовать тип char с ограничением в 7 символов |
| Person. passport\_unit | Кем выдан паспорт | varchar(100) | Можно использовать тип данных varchar с ограничением длины в 100 символов. varchar(n) позволяет хранить строки переменной длины, что и требуется, поскольку данные пользователей могут иметь разную длину |
| Person. passport\_birthday | Дата рождения заемщика | date | Позволяет хранить дату в формате YYYY-MM-DD. В отличие от timestamp занимает всего 4 байта |
| Person. passport\_placeOfBirth | Место рождения замещика | varchar(100) | Можно использовать тип данных varchar с ограничением длины в 100 символов. varchar(n) позволяет хранить строки переменной длины, что и требуется, поскольку данные пользователей могут иметь разную длину |
| Person. passport\_place | Регион регистрации заемщика | varchar(100) | Можно использовать тип данных varchar с ограничением длины в 100 символов. varchar(n) позволяет хранить строки переменной длины, что и требуется, поскольку данные пользователей могут иметь разную длину |
| Person. work\_region | Регион работы заемщика | varchar(100) | Можно использовать тип данных varchar с ограничением длины в 100 символов. varchar(n) позволяет хранить строки переменной длины, что и требуется, поскольку данные пользователей могут иметь разную длину |
| Person. work\_organization | Название организации заемщика | varchar(100) | Можно использовать тип данных varchar с ограничением длины в 100 символов. varchar(n) позволяет хранить строки переменной длины, что и требуется, поскольку данные пользователей могут иметь разную длину |
| Person. work\_inn | ИНН организации заемщика | char(10) | ИНН – это фиксированное значение, состоящее из 10 цифр. Здесь целесообразнее использовать тип char с ограничением в 10 символов |
| Person. work\_position | Должность заемщика | varchar(100) | Можно использовать тип данных varchar с ограничением длины в 100 символов. varchar(n) позволяет хранить строки переменной длины, что и требуется, поскольку данные пользователей могут иметь разную длину |
| Person. work\_salary | Заработная плата заемщика | real | Можно использовать тип данных real, который позволяет хранить числа с плавающей точкой. Для значений данного типа выделяется 4 байта. |
| Person. work\_startofDay | Дата начала работы заемщика | date | Позволяет хранить дату в формате YYYY-MM-DD. В отличие от timestamp занимает всего 4 байта |

Таблица 2. Сущность Query

|  |  |  |  |
| --- | --- | --- | --- |
| **Сущность-атрибут** | **Описание** | **Тип данных** | **Причина выбора** |
| Query.id | Идентификатор заявки | integer | Тип данных integer позволяет хранить целые числа с верхним пределом 2147483647, занимает 4 байта. |
| Query.id\_name | Наименование продукта | smallint | Идентификатору сервиса можно назначить тип smallint, поскольку данный тип позволяет хранить числа до верхнего предела в 32767. В отличие от integer занимает 2 байта. |
| Query.id\_purpose | Цель продукта | smallint | Идентификатору сервиса можно назначить тип smallint, поскольку данный тип позволяет хранить числа до верхнего предела в 32767. В отличие от integer занимает 2 байта. |
| Query.summ | Сумма | real | Для суммы кредита можно использовать тип данных real, который позволяет хранить числа с плавающей точкой. Для значений данного типа выделяется 4 байта. |
| Query.rate | Размер ставки | real | Для размера ставки можно использовать тип данных real, который позволяет хранить числа с плавающей точкой. Для значений данного типа выделяется 4 байта. |
| Query.period | Срок кредита | smallint | Для кредитного срока можно использовать тип данных smallint, который позволяет хранить целые числа с верхним пределом 32767. Для значений данного типа выделяется 2 байта. |
| Query.content\_to\_per\_data | Согласие на обработку персональных данных | boolean | Согласие на обработку персональных данных может принимать два значения – да(true) и нет(false) |
| Query.date | Дата формирование заявки | timestamp | Использование timestamp целесообразнее поскольку позволяет сохранить дату и время создания заявки и в последующем обеспечивает удобность сортировки |

Таблица 3. Сущность getQuery

|  |  |  |  |
| --- | --- | --- | --- |
| **Сущность-атрибут** | **Описание** | **Тип данных** | **Причина выбора** |
| getQuery.id | Идентификатор оформления заявки | integer | Тип данных integer позволяет хранить целые числа с верхним пределом 2147483647, занимает 4 байта. |
| getQuery.id\_person | Идентификатор заемщика | integer | Тип данных integer позволяет хранить целые числа с верхним пределом 2147483647, занимает 4 байта. |
| getQuery.id\_query | Идентификатор заявки | integer | Тип данных integer позволяет хранить целые числа с верхним пределом 2147483647, занимает 4 байта. |

Таблица 4. Сущность getServices

|  |  |  |  |
| --- | --- | --- | --- |
| **Сущность-атрибут** | **Описание** | **Тип данных** | **Причина выбора** |
| getServices.id | Идентификатор включения дополнительных сервисов | integer | Тип данных integer позволяет хранить целые числа с верхним пределом 2147483647, занимает 4 байта. |
| getServices.id\_query | Идентификатор заявки | integer | Тип данных integer позволяет хранить целые числа с верхним пределом 2147483647, занимает 4 байта. |
| getServices.id\_service | Идентификатор сервиса | integer | Тип данных integer позволяет хранить целые числа с верхним пределом 2147483647, занимает 4 байта. |
| getServices.price | Стоимость сервиса | real | Можно использовать тип данных real, который позволяет хранить числа с плавающей точкой. Для значений данного типа выделяется 4 байта. |

Таблица 5. Сущность Services

|  |  |  |  |
| --- | --- | --- | --- |
| **Сущность-атрибут** | **Описание** | **Тип данных** | **Причина выбора** |
| Services.id | Идентификатор сервиса | smallint | Идентификатору сервиса можно назначить тип smallint, поскольку данный тип позволяет хранить числа до верхнего предела в 32767. В отличие от integer занимает 2 байта. |
| Services.name | Название сервиса | varchar(100) | Можно использовать тип данных varchar с ограничением длины в 100 символов. varchar(n) позволяет хранить строки переменной длины, что и требуется, поскольку названия сервисов могут иметь разную длину |

Таблица 6. Сущность QueryName

|  |  |  |  |
| --- | --- | --- | --- |
| **Сущность-атрибут** | **Описание** | **Тип данных** | **Причина выбора** |
| QueryName.id | Идентификатор наименования продукта | smallint | Идентификатору сервиса можно назначить тип smallint, поскольку данный тип позволяет хранить числа до верхнего предела в 32767. В отличие от integer занимает 2 байта. |
| QueryName.name | Цель продукта | varchar(100) | Можно использовать тип данных varchar с ограничением длины в 100 символов. varchar(n) позволяет хранить строки переменной длины, что и требуется, поскольку наименования продуктов могут иметь разную длину |

Таблица 7. Сущность QueryPurpose

|  |  |  |  |
| --- | --- | --- | --- |
| **Сущность-атрибут** | **Описание** | **Тип данных** | **Причина выбора** |
| QueryPurpose.id | Идентификатор наименования цели | smallint | Идентификатору сервиса можно назначить тип smallint, поскольку данный тип позволяет хранить числа до верхнего предела в 32767. В отличие от integer занимает 2 байта. |
| QueryPurpose.purpose | Цель продукта | varchar(100) | Можно использовать тип данных varchar с ограничением длины в 100 символов. varchar(n) позволяет хранить строки переменной длины, что и требуется, поскольку цели могут иметь разную длину |

*Типы данных указаны в соответствии с документацией, представленной на сайте* [*PostgresPro*](https://postgrespro.ru/docs/postgresql/15/datatype)

**Задание 2**

1. SQL-запрос написан для СУБД PostgreSQL 16.2

*SELECT COUNT(\*) AS "count", "QueryName"."name" FROM "Query"*

*JOIN "QueryName" ON "Query"."id\_name" = "QueryName"."id"*

*WHERE date\_part('year', "date") = date\_part('year', CURRENT\_DATE)*

*GROUP BY "QueryName"."name"*

*ORDER BY "count" DESC*

*LIMIT 1;*

**Задание 3**

1. Алгоритм расчета аннуитетных платежей:

Для расчета аннуитетных платеже был создан класс Annuitent. В конструктор класса нужно передать три свойства – сумму кредита в рублях, ставку в процентах и срок кредита.

В функции getMonthPay рассчитывается сумма аннуитетного платежа за месяц по формуле:

*Сумма кредита × ( (Годовая ставка / 100 / 12) / (1 - (1 + (Годовая ставка / 100 / 12) ) - Количество месяцев ) )*

В функции getPercentSum подсчитывается долг по процентам

Результирующий подсчет и вывод в консоль проводится в функции calculate.

1. Функция на JS:

class Annuitent {

    constructor(sum, rate, period) {

*this*.summ = sum;                            // Сумма кредита

*this*.rate = rate / 100;                     // Ставка в процентах / 100

*this*.period = period;                       //Срок кредита в месяцах

*this*.monthRate = *this*.rate / 12;            //процентная ставка в месяц, равная годовой ставке, поделённой на 12

*this*.countDayYear = 365                     //Количество дней в году

*this*.persentDay = *this*.countDayYear / 12      //Количество дней, за которые начисляются проценты

    }

    // Ежемесячный платеж

    getMonthPay() {

        const monthPayK = (*this*.monthRate \* ((1 + *this*.monthRate) \*\* *this*.period)) / (((1 + *this*.monthRate) \*\* *this*.period) - 1)

        const monthPay = monthPayK \* *this*.summ

        return monthPay

    }

    //Окргуляет до двух знаков после запятой

    roundup(numb) {

        return Math.round(numb \* 100) / 100

    }

    //Сумма переплаты

    getbankPay(monthPay) {

        return (monthPay \* *this*.period) - *this*.summ

    }

    //Долг по процентам

    getPersentSumm(summ) {

        return *this*.roundup(summ \* (*this*.persentDay \* *this*.rate / *this*.countDayYear))

    }

    //Расчёт аннуитетного платежа

    calculate() {

        const monthPay = *this*.roundup(*this*.getMonthPay())

        let year = 1

        let summ = *this*.summ

        console.log('Месяц\tЕжемесячный платеж\tОсновной долг\tДолг по процентам\tОстаток основного долга')

        let start = true

        let mainPay;

        let su = 0;

        while (summ > 0) {

            if (start) {

                mainPay = *this*.roundup(monthPay - *this*.getPersentSumm(summ))

                console.log(year, '\t', monthPay, '\t\t', mainPay, '\t', *this*.getPersentSumm(summ), '\t\t', summ - mainPay)

            }

            else {

                mainPay = *this*.roundup(monthPay - *this*.getPersentSumm(su))

                console.log(year, '\t', monthPay, '\t\t', mainPay, '\t', *this*.getPersentSumm(su), '\t\t', su - mainPay)

            }

            su = summ - mainPay

            year += 1

            summ -= mainPay

            start = false

        }

    }

}

let summ1 = 100000;     // Пример 1. Сумма кредита

const rate1 = 13        // Пример 1. Ставка в процентах

const period1 = 12      // Пример 1. Срок кредита в месяц

const example1 = new Annuitent(summ1, rate1, period1)

console.log(example1.calculate())

let summ2 = 350000;     // Пример 2. Сумма кредита

const rate2 = 12        // Пример 2. Ставка в процентах

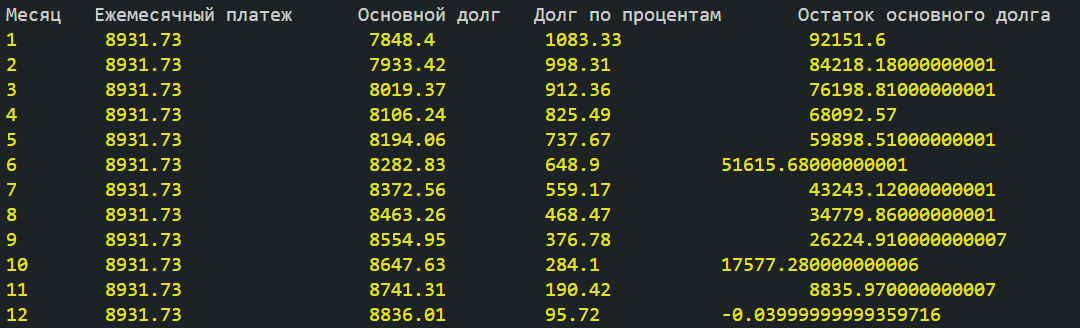
const period2 = 24      // Пример 2. Срок кредита в месяц

const example2 = new Annuitent(summ2, rate2, period2)

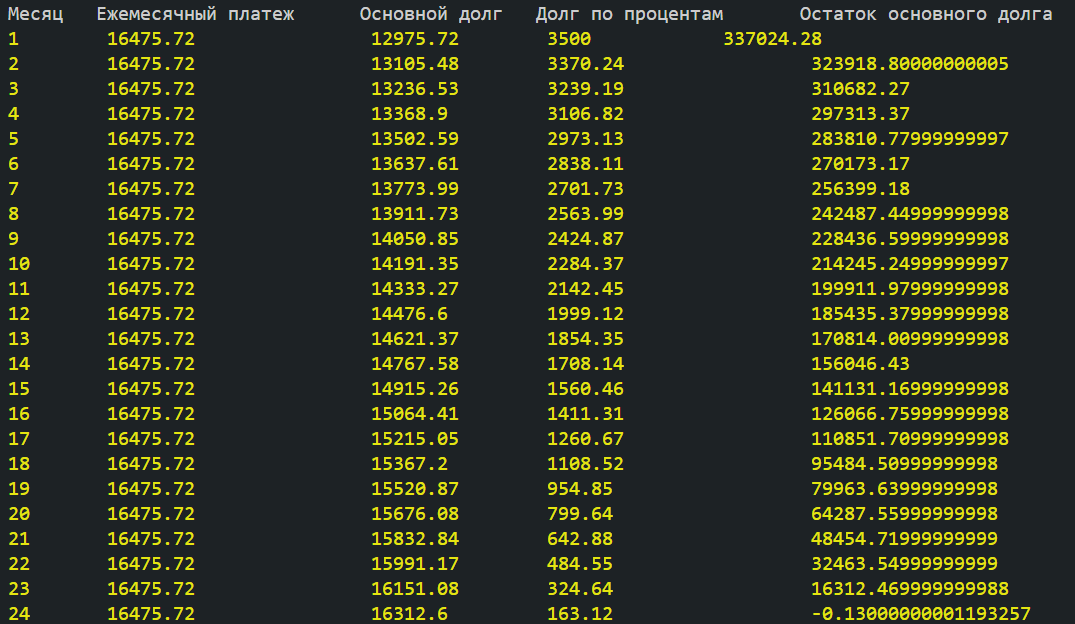
console.log(example2.calculate())

1. Тестирование алгоритма:

|  |  |  |
| --- | --- | --- |
| Сумма кредита, руб | Ставка, % | Срок, месяцев |
| 100000 | 13 | 12 |



|  |  |  |
| --- | --- | --- |
| Сумма кредита, руб | Ставка, % | Срок, месяцев |
| 350000 | 12 | 24 |



Данный алгоритм расположен в репозитории на GitHub и доступен по ссылке. Для запуска алгоритма используйте Node.js – *node index.js*