fМИНОБРНАУКИ РОССИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ

ВЫСШЕГО ОБРАЗОВАНИЯ

“ВОРОНЕЖСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ”

ФГБОУ ВО «ВГУ»

Факультет компьютерных наук

Кафедра программирования и информационных технологий

Курсовой проект

Генератор анонимизированных баз данных

09.03.04 Программная инженерия

Информационные системы и сетевые технологии

Зав. кафедрой_____ С. Д. Махортов, д.ф, -м.н., профессор

Обучающийся_____ М. А. Ячный, 3 курс, очная

Руководитель_____ Н.К. Самойлов, ст. преподаватель

Воронеж 2025

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Современный мир информационных технологий характеризуется стремительным развитием программного обеспечения и увеличением объемов обрабатываемых данных. В условиях ужесточения требований к защите персональной информации, таких как Федеральный закон "О персональных данных" в России[1] и Общий регламент по защите данных в ЕС[2], анонимизация данных становится критически важным процессом. Она позволяет использовать реальные данные в разработке и тестировании программного обеспечения, минимизируя риски нарушения конфиденциальности.

Анонимизация данных — это процесс преобразования информации таким образом, чтобы исключить возможность идентификации личности или восстановления исходных данных. В разработке программного обеспечения она применяется для тестирования на реалистичных наборах данных без раскрытия персональных сведений. Это особенно важно в таких сферах, как здравоохранение, банковское дело и электронная коммерция, где утечка данных может привести к серьезным последствиям.

Цель данной курсовой работы — разработка генератора анонимизированных реляционных баз данных, который позволит создавать реалистичные, но при этом безопасные с точки зрения конфиденциальности наборы данных для использования в разработке и тестировании ПО.

1 Постановка задачи

Целью курсовой работы является создание серверного приложения для генерации анонимизированных локальных баз данных, которое позволяет формировать тестовые данные, соответствующие заданной структуре, с учётом основных типов данных и ограничений целостности. Приложение должно сохранять реалистичность данных и их пригодность для использования в разработке и тестировании ПО.

Приложение должно обеспечивать возможность работы со следующими типами данных:

— Целые числа;

— Числа с плавающей точкой;

— Строки;

— Даты;

— Числовые интервалы;

— Интервалы дат.

Кроме того, приложение должно учитывать возможность наличия пустого значения (NULL) в полях реляционных таблиц.

Приложение должно учитывать следующие ограничения целостности данных:

— Первичный ключ;

— Внешний ключ;

— Уникальный ключ.

Приложение должно обеспечивать возможность использования в качестве типа данных первичного ключа следующих типов данных:

— Целое число;

— UUID;

— ObjectID.

Приложение должно обеспечивать возможность работы с двумя типами кардинальности связей между реляционными таблицами, а именно «Один к одному» и «Один ко многим». Также должна быть обеспечена возможность задания правила того, является ли обязательным наличие сущности (сущностей) с левой и с правой стороны связи. В случае связи «Один ко многим» приложение должно позволить задать правило, согласно которому будет определяться количество сущностей с правой стороны связи.

Приложение должно поддерживать возможность многопоточного заполнения базы данных, т.е. одновременного (параллельного) заполнения нескольких таблиц.

Список таблиц базы данных, типы данных столбцов таблиц, ограничения целостности, кардинальность связей между таблицами, правила, согласно которым будут генерироваться значения полей таблиц, передаётся в REST-запросе в одном объекте формата JSON.

2 Анализ предметной области

2.1 Выбор используемых технологий

Для реализации генератора анонимизированных баз данных был выбран язык Java. Данный выбор обусловлен рядом технических и практических преимуществ, которые делают эту комбинацию оптимальной для решения поставленной задачи. Java, как строго типизированный объектно-ориентированный язык с мощной экосистемой, предоставляет все необходимые инструменты для работы с базами данных на низком уровне через JDBC[3].

Использование JDBC (Java Database Connectivity) обеспечивает прямой доступ к функциональности СУБД, позволяя работать с отдельными столбцами таблиц и вручную накладывать ограничения на них.

Приложение поддерживает возможность работы с несколькими СУБД, однако в качестве примера была выбрана PostgreSQL благодаря развитой системе типов данных и мощной поддержке ограничений целостности[4], несмотря на простоту синтаксиса запросов.

Для работы с подключениями к базе данных используется Hikari Connection Pool — это наиболее производительный и надежный пул соединений, который значительно превосходит альтернативные решения по скорости работы и эффективности управления ресурсами.

Для реализации серверных функций используется платформа Spring Boot, представляющая собой оптимальную платформу для разработки серверного приложения благодаря своей комплексной экосистеме и архитектурным преимуществам, и обеспечивающая готовую инфраструктуру для создания современных веб-сервисов.

2.2 Подходы к анонимизации данных

Современные подходы к анонимизации данных направлены на решение задачи защиты конфиденциальности информации. В области защиты данных применяются различные техники анонимизации, каждая из которых имеет свои особенности.

Обобщение предполагает замену точных значений на диапазоны или категории (например, замена значения возраста человека с «24» на «20-30»), что полезно для аналитических отчётов, но мало применимо в тестировании, где требуются конкретные значения.

Псевдонимизация использует замену идентификаторов на псевдонимы с использованием хеширования или таблиц соответствия, что сохраняет риски для конфиденциальности, т.к. позволяет восстановить исходные данные при наличии ключа.

Шифрование данных хотя и обеспечивает высокий уровень защиты, но делает данные непригодными для непосредственного тестирования без этапа дешифрования.

Частичное удаление полей может нарушить целостность базы данных и логику работы приложения.

На этом фоне маскировка данных выделяется как наиболее сбалансированный подход. Маскировка данных — это процесс замены оригинальных значений на искусственно сгенерированные, которые сохраняют все формальные характеристики исходных данных, но при этом исключают возможность идентификации личности. Главное преимущество маскировки заключается в её способности сохранять структуру данных и их пригодность для тестирования. В отличие от шифрования или удаления, этот метод не нарушает целостность базы данных и позволяет разработчикам работать с информацией, которая по всем формальным признакам соответствует реальной. Это особенно важно при тестировании функционала,

чувствительного к типам данных и форматам — валидации форм, обработке транзакций или генерации отчётов.

С точки зрения безопасности маскировка обеспечивает необратимое преобразование данных, что соответствует строгим требованиям современных стандартов защиты информации.

2.3 Способы генерации интервальных значений

При генерации тестовых данных, включающих временные периоды или числовые диапазоны, особую важность приобретает корректное моделирование отношений между интервалами. Фундаментальную основу для работы с такими отношениями составляет интервальная алгебра Аллена – формальная система, описывающая все возможные варианты взаимного расположения двух временных или числовых интервалов[5]. Алгебра Аллена определяет 13 базовых отношений между двумя интервалами A и B, которые полностью исчерпывают все возможные варианты их взаимного расположения на временной или числовой оси, однако при детальном анализе обнаруживается, что некоторые из этих отношений являются либо зеркальными отображениями друг друга, либо могут быть выражены через комбинации других. Для задач генерации тестовых данных мы можем существенно сократить базовый набор, оставив только 6 ключевых отношений, сохранив при этом всю выразительную мощность алгебры:

— Перекрытие (A перекрывает B): начало A до начала B, конец A после начала B, но до конца B;

— Обратное перекрытие (B перекрывается A): зеркальное отношение к перекрытию;

— Следование (A встречает B): конец A точно совпадает с началом B;

— Обратное следование (B встречен A): зеркальное отношение к непосредственному следованию;

— Включение (A во время B): A полностью содержится внутри B;

— Обратное включение (B содержит A) : Начало B до начала A, конец B после конца A.

Визуальное представление используемых в системе правил приведено на рисунке 1.

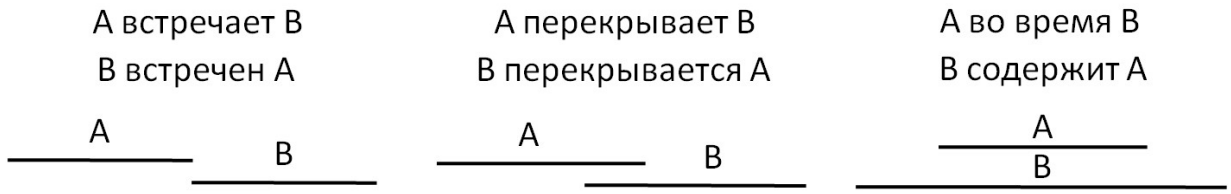

Рисунок 1 – Визуализация правил алгебры Аллена

На рисунке 2 приведён пример тождественности одного из правил алгебры Аллена комбинации двух других отношений. Здесь интервал B фиксирован и имеет начало в точке 0, конец в точке 10. Задаваемое правило можно интерпретировать следующим образом: интервал A обязательно имеет начало в точке 0, а его концом является любая точка от 0 до 10. Данное правило возможно задать двумя способами: одним отношением «A начинает B», либо тождественной ему комбинацией отношений «A во время B» и «A встречен C», где C – любой интервал, конец которого совпадает с началом B.

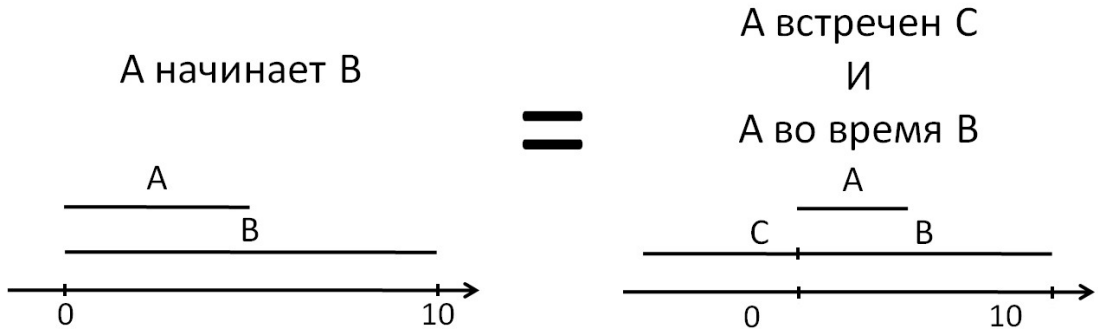

Рисунок 2 – Тождественность правил алгебры Аллена

3 Реализация

3.1 Структура приложения

В рамках практической реализации генератора анонимизированных баз данных в системе можно выделить несколько основных компонентов. Рисунок 3 иллюстрирует архитектуру приложения.

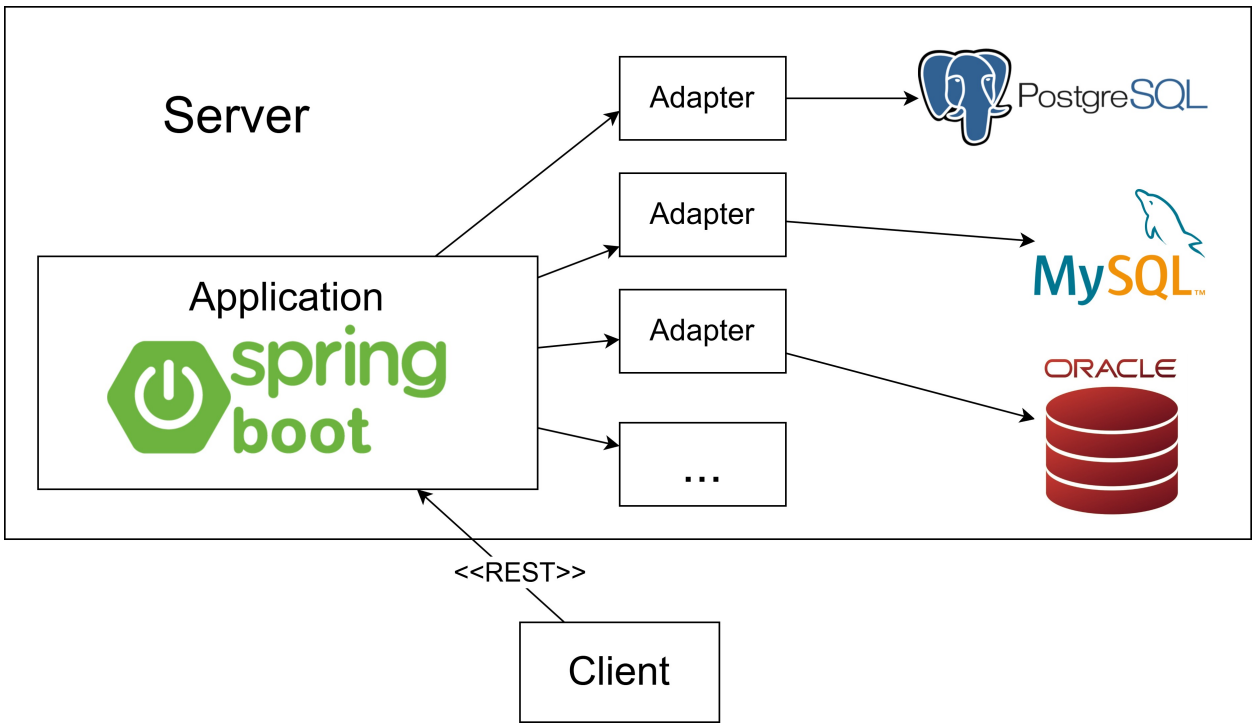

Рисунок 3 – Архитектура приложения

Приложение, развёрнутое на сервере, представляет собой центральное звено архитектуры. Оно принимает запросы от клиента и взаимодействует с базой данных через адаптер.

Клиент в данном случае является «тонким», т.е. все задачи по обработке запросов и вычислениям производятся на сервере, единственная задача клиента – выполнение REST-запросов к серверу.

Адаптеры являются посредниками между сервером и системами управления базами данных. Наличие адаптеров обеспечивает расширяемость приложения и возможность подключения практически любой СУБД

посредством написания конкретного адаптера под неё, который будет учитывать синтаксис запросов данной СУБД, и какие классы драйверов необходимы для подключения к ней. Адаптеры предоставляют возможность выполнения следующих типов запросов к базе данных:

— Удаление таблицы

— Создание таблицы

— Вставка записей в таблицу

— Обновление записей в таблице

— Наложение ограничений на столбцы таблицы

3.2 Анализ исходных данных

В рамках практической реализации генератора анонимизированных баз данных алгоритм работы программы строится на последовательном выполнении трех ключевых этапов, обеспечивающих формирование корректных и согласованных тестовых данных. Первый этап предполагает анализ исходных данных из переданного через REST-запрос объекта. На этом этапе программа анализирует типы данных, требуемые распределения, допустимые диапазоны значений, кратность связей между таблицами.

В поле databaseSchema переданного через запрос JSON-объекта должен содержаться объект класса DatabaseSchema, данный класс является корневым элементом модели, описывающей схему базы данных и содержит список таблиц, представленных объектами класса Table. Класс Table содержит поле name строкового типа – имя таблицы, значение которого должно быть уникальным среди всех таблиц, и список столбцов, представленных объектами класса Column. Данный класс содержит строковое поле name – имя столбца, значение которого должно быть уникальным в пределах таблицы, а также поле type строкового типа, описывающее тип данных столбца. Структура и отношения вышеописанных классов приведены диаграммой классов на рисунке 4.

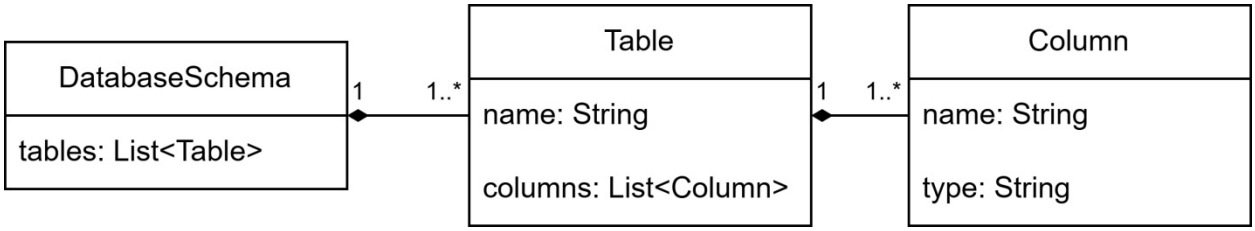

Рисунок 4 – Диаграмма классов схемы базы данных

В поле rules переданного в запросе JSON-объекта содержится экземпляр класса RuleSet. В нём задаются правила, по которым будут генерироваться значения в столбцах, не являющихся первичными или внешними ключами. В данном классе для каждого из допустимых в приложении типа данных содержится список правил, касающихся правил генерации значений в столбцах этого типа данных.

Правила генерации значений каждого типа содержатся в экземплярах соответствующего класса, реализующего интерфейс Rule. Данный интерфейс имеет методы getTableName() и getColumnName() для доступа к именам таблицы и столбца, правила генерации значений в котором описывает правило. Также в интерфейсе имеется метод getNullChance() для удобного доступа к значению вероятности того, что очередное сгенерированное значение будет пустым. Кроме того, в интерфейсе содержится метод toGenerator(), предназначенный для удобного и быстрого преобразования правила в готовый генератор значений данного типа. На рисунке 5 представлена диаграмма классов для набора правил генерации значений. Классы для правил конкретных типов будут подробно описаны далее в этом разделе.

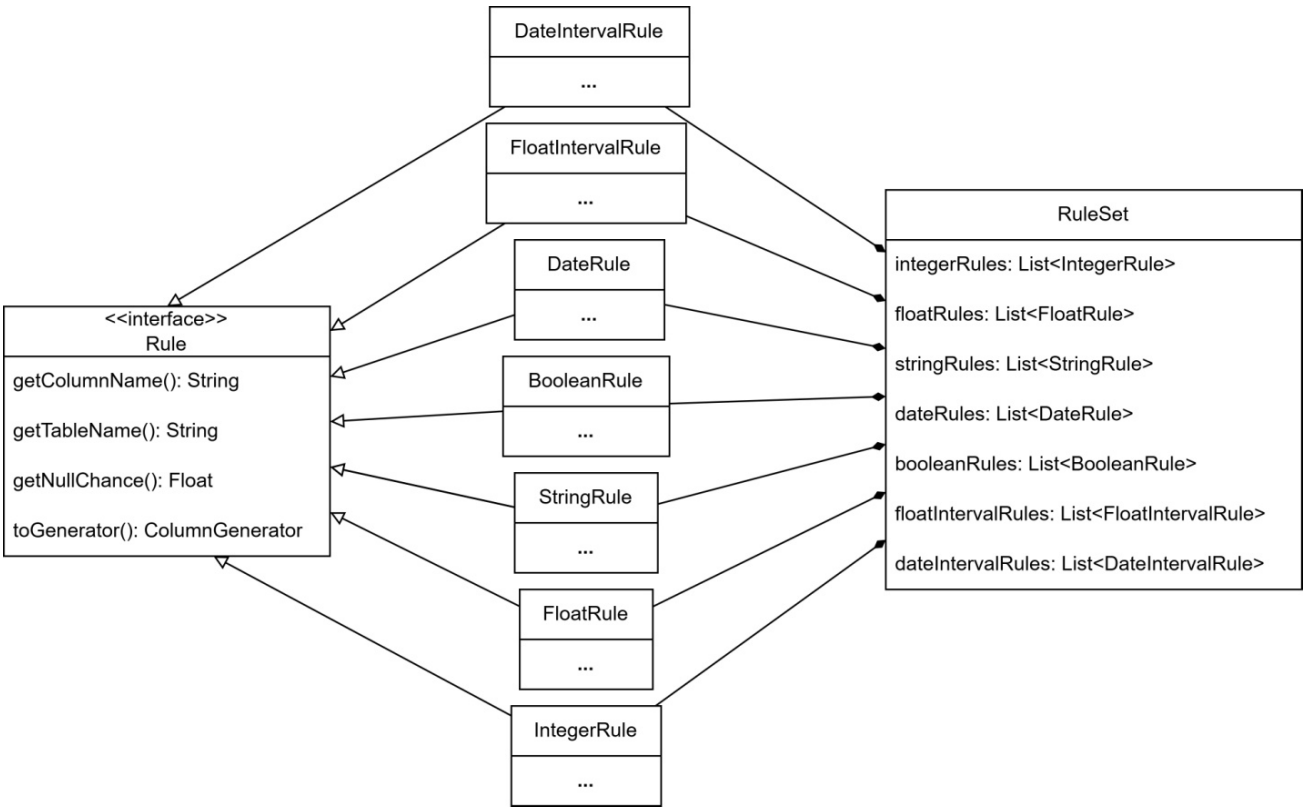

Рисунок 5 – Диаграмма классов набора правил генерации

Важно отметить, что все классы правил генерации данных содержат поля строкового типа columnName и tableName, а также поле типа числа с плавающей точкой nullChance. Значения, хранящиеся в этих полях, возвращаются в соответствующих трёх методах, унаследованных от интерфейса Rule. Далее при описании классов правил для конкретных типов данных эти три поля не будут упоминаться.

На рисунке 6 приведена диаграмма классов для правил генерации численных типов данных: IntegerRule и FloatRule. В обоих классах содержится поле distributionType, задающее тип распределения (дискретного в случае IntegerRule и непрерывного в случае FloatRule), а также поле params, являющееся массивом чисел с плавающей точкой и задающее параметры случайного распределения [6].

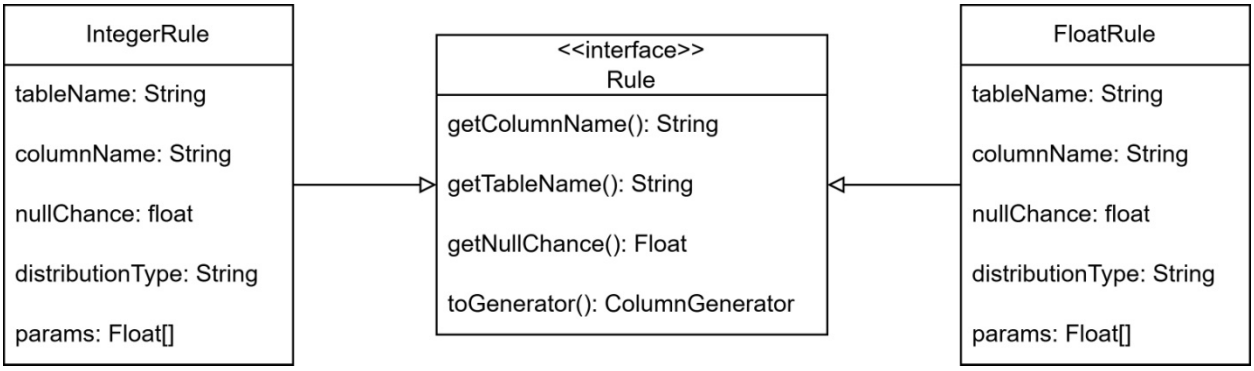

Рисунок 6 – Диаграмма классов для правил целых численных значений

Правило генерации строковых данных – StringRule – содержит в себе список экземпляров класса WordRule, каждый элемент которого является правилом для генерации отдельного слова, данный список хранится в поле wordRules. Также внутри класса имеется поле строкового типа separator, задающее, какой последовательностью символов будут разделены сгенерированные слова.

Класс WordRule задаёт правило генерации отдельного слова и имеет поле allowedCharacters, представляющее собой набор символов, разрешённых для использования в сгенерированных значениях. Полями distributionType и distributionParams, по аналогии с классом IntegerRule, задаётся дискретное распределение для генерации случайного значения длины слова. На рисунке 7 приведена диаграмма классов для правила генерации строковых значений.

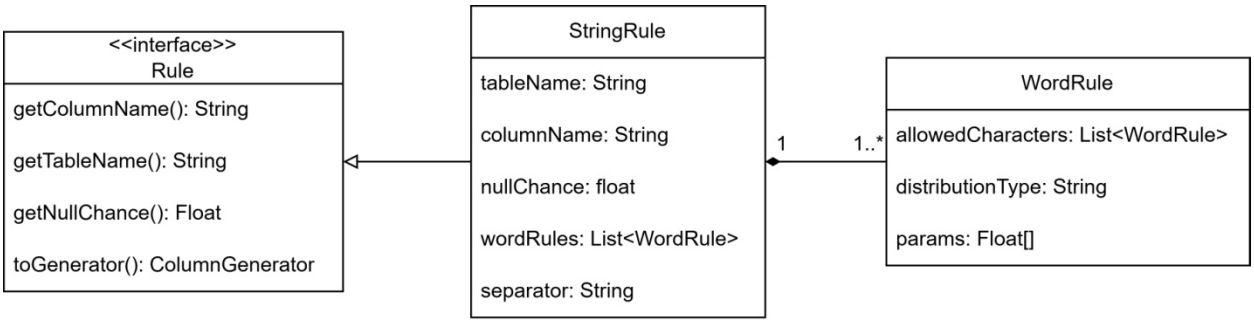

Рисунок 7 – Диаграмма классов для правил генерации строковых значений

Правило генерации значения даты описывается классом DateRule. Внутри содержатся два поля startDate и endDate, задающие границы интервала, внутри которого будет генерироваться случайная дата. В данных полях записаны строки формате ISO-8601[7]. На рисунке 8 представлена диаграмма классов для данного вида правила.

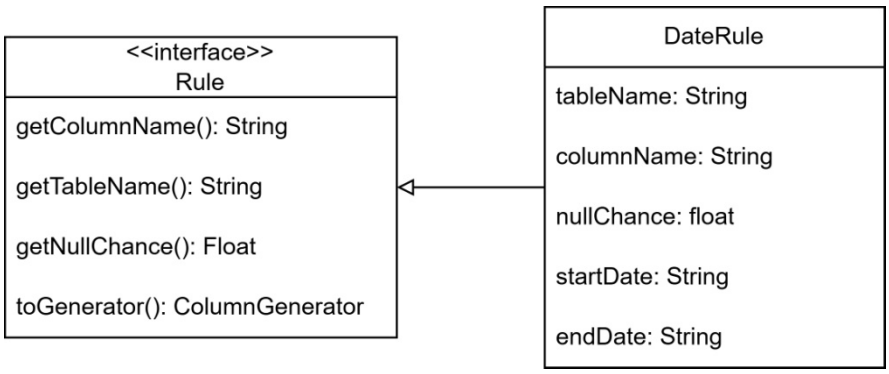

Рисунок 8 – Диаграмма классов для правил генерации дат

Правила генерации значений интервалов чисел с плавающей точкой задаются в классе FloatIntervalRule. В классе имеется поле relations – список экземпляров класса FloatIntervalRelation, каждый из которых определяет то, в каком отношении с описанным в полях start и end (в классе FloatIntervalRule данные два поля имеют тип числа с плавающей точкой, в классе DateIntervalRule – строковый тип) интервалом обязан быть генерируемый интервал. Отношение является одним из правил интервальной алгебры Аллена. Каждое из содержащихся в поле relations правил должно выполняться для сгенерированного интервала одновременно. Диаграмма классов для правил генерации интервальных значений изображена на рисунке 9.

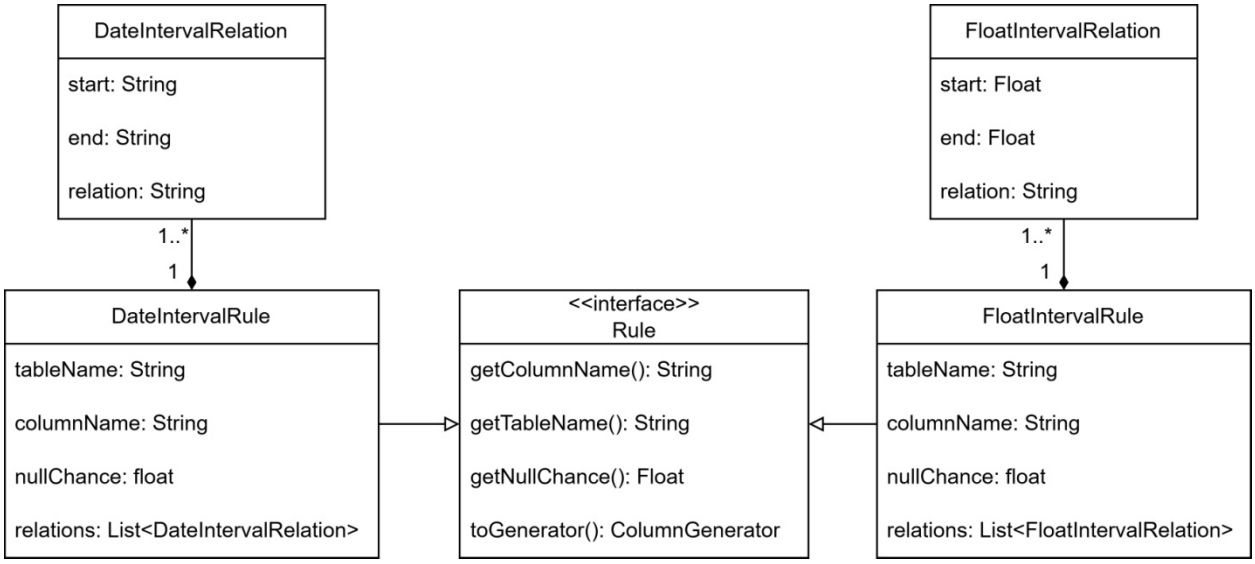

Рисунок 9 – Диаграмма классов для правил генерации интервалов

Поле constraintSet переданного JSON-объекта содержит экземпляр класса ConstraintSet и описывает, к каким столбцам таблиц необходимо применить ограничения.

Ограничения первичного ключа задаются в поле primaryKeys, являющимся списком объектов класса PrimaryKey. Каждый из них содержит строковые поля columnName и tableName, указывающие, к какой таблице и столбцу необходимо применить ограничение.

Поле foreignKeys содержит список объектов класса ForeignKey и указывает, к каким таблицам и столбцам необходимо применить ограничение внешнего ключа. В данном классе имеются поля строкового типа sourceTableName и sourceColumnName, указывающие на столбец, являющийся ссылкой, а также targetTableName и targetColumnName, определяющие столбец, на который указывает данная ссылка. Кратность связи задаётся полями sourceDistributionType и sourceDistributionParams в виде дискретного случайного распределения. Поле sourceZeroChance задаёт, для какой доли записей с левой стороны связи не окажется сущностей с правой стороны, т.е. ссылка будет содержать пустое значение. Поле targetZeroChance определяет, на какую долю записей с правой стороны не будут указывать никакие записи с левой стороны.

Таким способом можно описать не только связь типа «Многие к одному», но и связь типа «Один к одному», указав в качестве случайного распределения равномерное с параметрами 1 и 2 (оно всегда будет генерировать число 1).

Ограничения уникальности описаны в поле uniques, являющемся списком объектов класса Unique. Внутри данного класса содержатся только поля columnName и tableName, указывающие на таблицу и столбец, к которому необходимо применить ограничение.

Диаграмма классов, описывающая набор ограничений, приведена на рисунке 10.

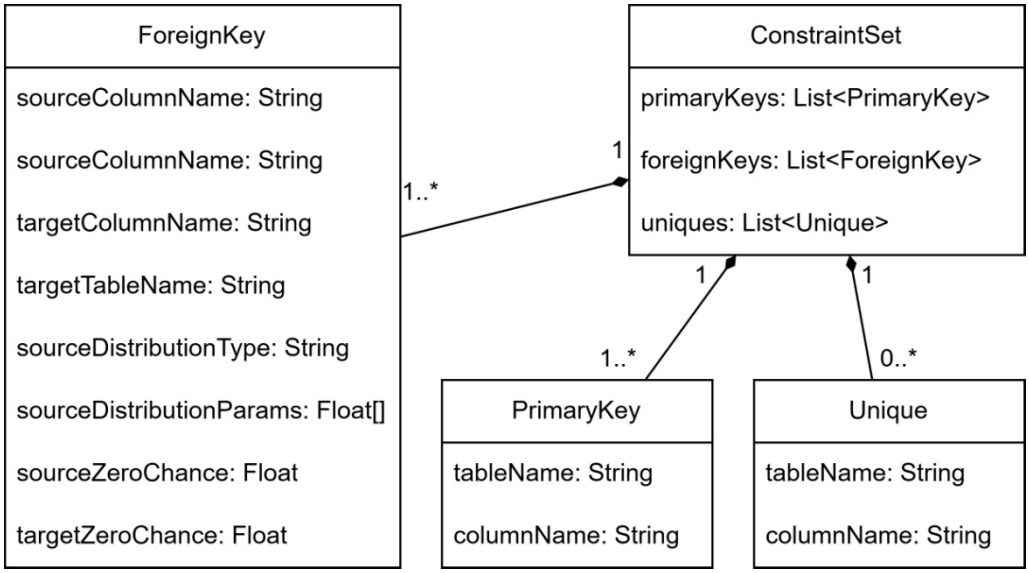

Рисунок 10 – Диаграмма классов для набора ограничений

Ограничения обязательных столбцов (NOT NULL) задаются согласно данным, описанным экземпляре класса RuleSet: ограничение накладывается на каждый столбец, которому соответствует правило, в поле nullChance которого указано значение 0.

3.3 Создание генераторов таблиц

Второй этап алгоритма работы приложения заключается в создании генераторов на основе проанализированных на первом этапе исходных данных, и затем непосредственной генерации записей в таблицах, используя созданные генераторы.

Генерация данных осуществляется в классе DatabaseGenerator, экземпляр которого создаётся с использованием трёх объектов классов DatabaseSchema, RuleSet и ConstraintSet, полученных на предыдущем этапе алгоритма. Также в DatabaseGenerator при создании передаётся имя таблицы, с которой необходимо начать заполнение, и сколько записей необходимо сделать в данной таблице.

Перед процессом непосредственного заполнения таблиц данными необходимо выполнить несколько подготовительных шагов, первым из которых является создание экземпляров класса TableGenerator, каждый из которых отвечает за заполнение отдельной таблицы. Внутри класса TableGenerator находится два поля: columnGenerators – список объектов абстрактного класса ColumnGenerator, и primaryKeyGenerator – экземпляр интерфейса PrimaryKeyGenerator.

Класс ColumnGenerator отвечает за генерацию отдельного столбца, его объекты создаются с помощью принадлежащего интерфейсу Rule метода toGenerator(), в который передаётся булево значение unique, задающее, должны ли быть генерируемые в столбце значения уникальными. В классе реализован только один метод (все остальные являются абстрактными) getNextValues(), который генерирует готовые к вставке в таблицу значения в виде массива строк с учётом возможного ограничения уникальности. Абстрактный метод generateValues() генерирует очередной набор значений без учёта ограничения уникальности. Используется именно массив строк, т.к. генератор может отвечать за заполнение сразу нескольких столбцов (в данной работе примером являются генераторы интервалов).

Интерфейс PrimaryKeyGenerator имеет два метода: getColumnName() для получения имени заполняемого столбца и getNextValue() для получения очередного значения первичного ключа в строковом виде. Экземпляры интерфейса создаются с помощью класса PrimaryKeyGeneratorFactory, статический метод которого принимает на вход имя столбца и тип данных первичного ключа. Диаграмма классов для генератора базы данных приведена на рисунке 11. Необходимо отметить, что для класса DatabaseGenerator в данной диаграмме приведено неполное описание. Оставшиеся поля и методы класса будут описаны позже в данном разделе.

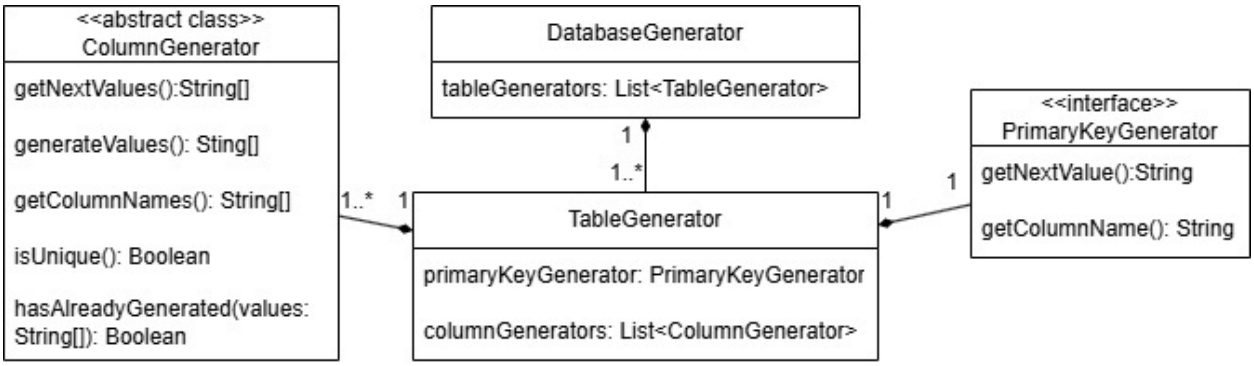

Рисунок 11 – Диаграмма классов для генераторов таблиц

Генерация интервалов сводится к определению двух значений, являющихся границами, в пределах которых может находиться начало интервала, и двух значений, в пределах которых может находиться конец интервала. Каждое из отношений алгебры Аллена порождает свой набор из данных значений.

Необходимо отметить, что некоторые отношения не задают одно или несколько из четырёх значений. Так, отношения «Перекрытие» и «Следование» не задают нижнюю границу для диапазона значений начальной точки интервала, а обратные им отношения не задают верхнюю границу для диапазона значений конечной точки интервала. Отношение «Обратное включение» не задаёт обе этих границы.

Все отношения, указанные в соответствующем экземпляре класса FloatIntervalRule или DateIntervalRule, связаны логическим умножением, т.е. должны выполняться одновременно. Для достижения этого из значений, задающих верхнюю границу диапазона значений начальной точки интервала, и значений, задающих верхнюю границу диапазона значений конечной точки интервала выбираются минимальные значения из всех, порождаемых отношениями. Для нижней границы начального диапазона и нижней границы конечного диапазона, соответственно, выбираются максимальные значения из всех, порождаемых отношениями.

3.4 Граф отношения таблиц и его обход

Для процесса генерации данных необходимо сформировать граф, который будет описывать отношения таблиц через внешние ключи. Граф, описанный в классе TableGraph, представляет собой список объектов класса TableGraphNode, каждый из которых содержит поле tableName строкового типа, а также поля children и parents типа List<RelationMapElement>. В первом содержится список таблиц, на которые указывают внешние ключи в таблице с именем tableName, во втором – список таблиц, внешние ключи которых ссылаются на таблицу с именем tableName. Класс RelationMapElement содержит поле foreignKeyName, обозначающее имя столбца с ограничением внешнего ключа, поле node типа TableGraphNode, указывающее на таблицу, с которой через этот внешний ключ связана исходная таблица, а также поле distribution, описывающее кратность связи. Кроме этого, имеются поля targetZeroChance и sourceZeroChance, значения которых взяты из соответствующих полей в соответствующем экземпляре класса ForeignKey в исходных данных.

Класс TableGraph заполняется на основе экземпляров классов DatabaseSchema и ConstraintSet, полученных при анализе исходных данных. Диаграмма классов для графа отношения таблиц приведена на рисунке 12.

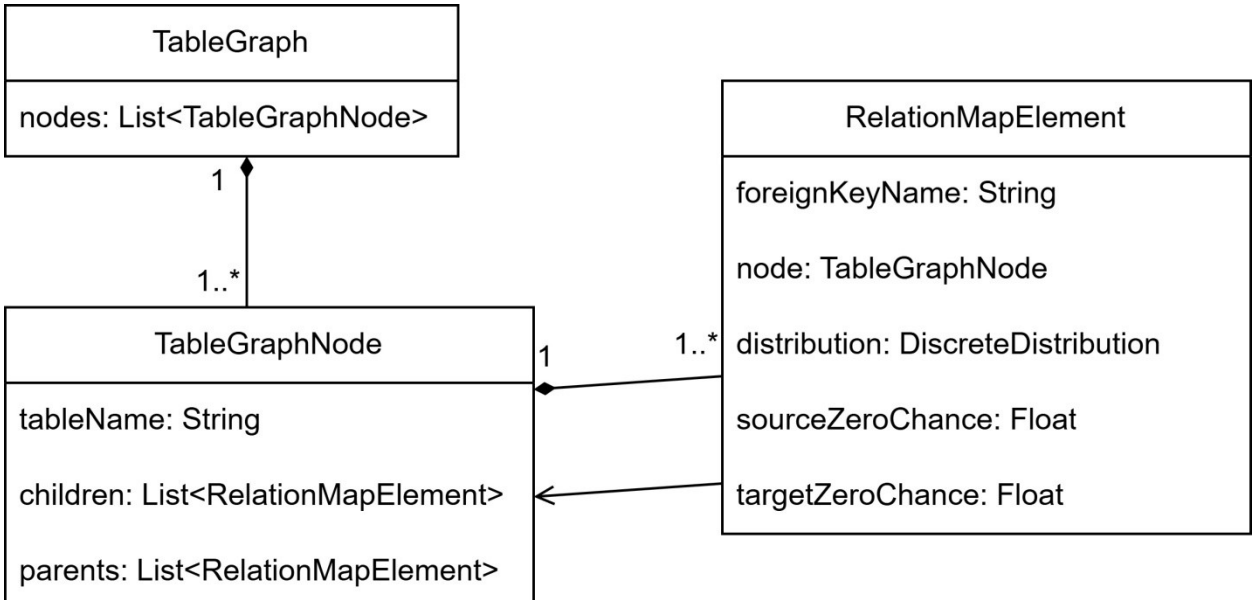

Рисунок 12 – Диаграмма классов для графа отношения таблиц

Рассмотрим пример графа отношения таблиц. На рисунке 13 приведена ER-диаграмма для базы данных, состоящей из 4 таблиц, связанных внешними ключами.

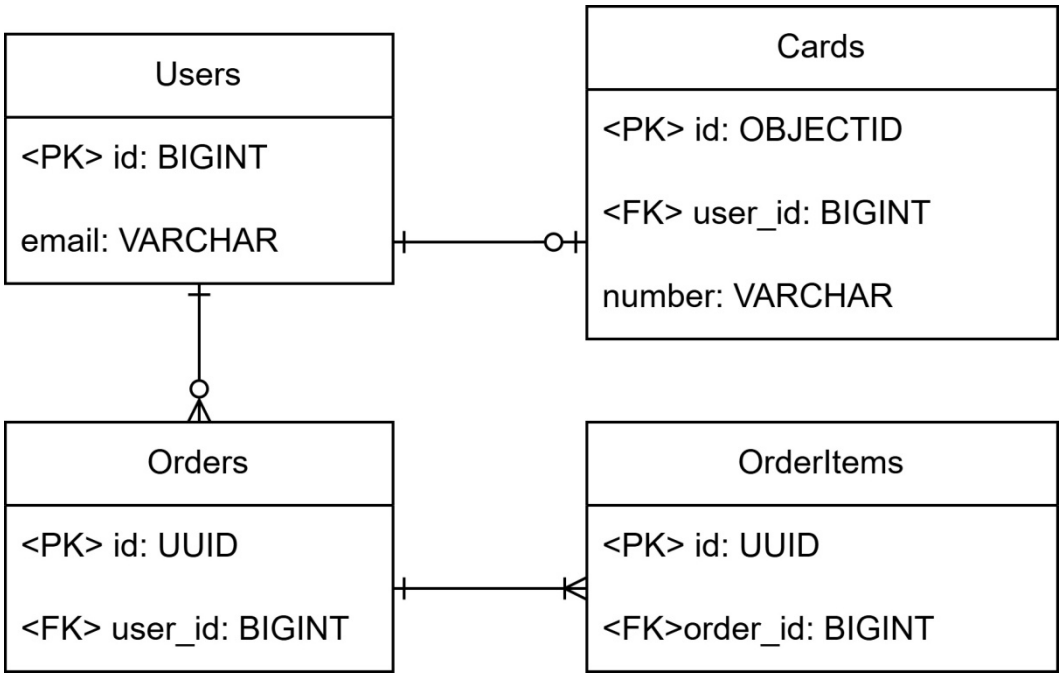

Рисунок 13 – ER-диаграмма для примера базы данных

Граф отношения таблиц, созданный на основе данного примера базы данных, приведён на рисунке 14.

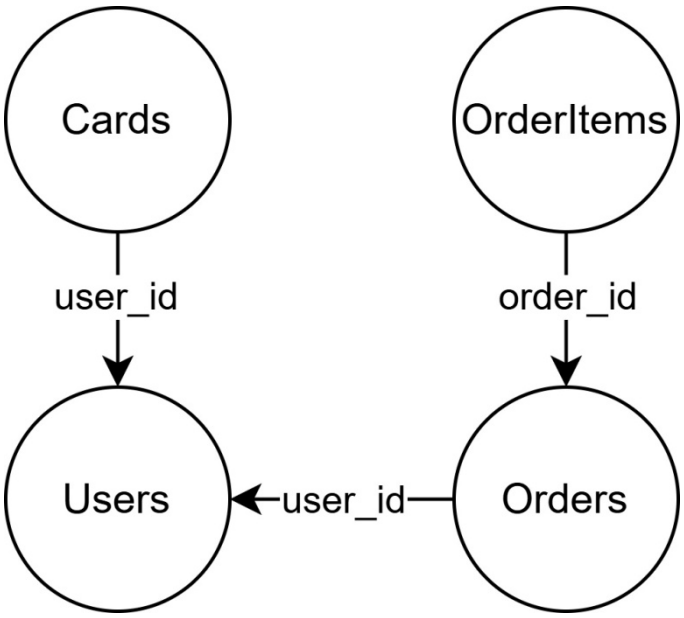

Рисунок 14 – Пример графа отношения таблиц

На выходе получается направленный граф, в узлах которого содержатся имена таблиц. Направления рёбер графа заданы следующим образом: связь идёт из таблицы, содержащей внешний ключ, в таблицу, на которую данных внешний ключ ссылается. В ребре хранится название внешнего ключа, через который связаны две таблицы, соединённые данным ребром. Алгоритм заполнения базы данных представляет собой обход созданного графа с началом таблице, имя которой передаётся в качестве одного из входных параметров. Введём следующие определения:

- Первая таблица – таблица, с которой начинается заполнение базы данных;
- Текущая таблица – таблица, заполнение которой производится на текущем шаге алгоритма;
- Родительская таблица – таблица, содержащая первичный ключ, ссылающийся на текущую таблицу;

— Дочерняя таблица – таблица, на которую ссылается один из внешних ключей текущей таблицы.

На приведённом выше примере графа таблицы Cards и Orders являются родительскими по отношению к таблице Users. Таблица Users является дочерней по отношению к этим двух таблицам. Таблица Order является дочерней по отношению к таблице OrderItems. Таблица OrderItems является родительской по отношению к таблице Orders.

Алгоритм генерации значений первой таблицы выполняется в методе fillFirstTable() класса DatabaseGenerator и состоит из следующих шагов:

— Генерация набора первичных ключей соответствующего типа;
— Создание случайной выборки из сгенерированных ключей, объём которой задаётся значением поля sourceZeroChance соответствующего экземпляра класса relationMapElement;
— Запуск заполнения каждой родительской таблицы в отдельном потоке с передачей в метод fillParentTable() созданной выборки значений первичных ключей;
— Заполнение первой таблицы с помощью генераторов столбцов
— Запуск заполнения каждой дочерней таблицы в отдельном потоке с передачей в метод fillChildTable() созданной выборки.

Диаграмма последовательностей, описывающая алгоритм заполнения первой таблицы, приведена на рисунке 15.

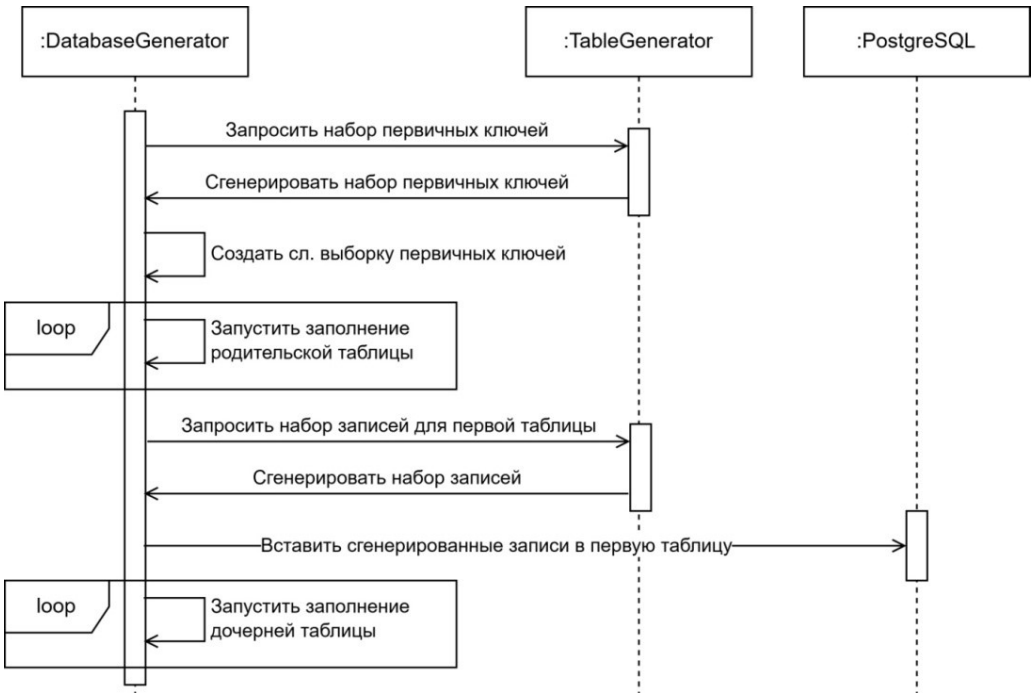

Рисунок 15 – Диаграмма последовательностей для метода fillFirstTable()

Алгоритм генерации значений родительской таблицы выполняется в методе fillParentTable() класса DatabaseGenerator и состоит из следующих шагов:

— Генерация набора первичных ключей соответствующего типа;

— Создание случайной выборки из сгенерированных ключей, объём которой задаётся значением поля sourceZeroChance соответствующего экземпляра класса relationMapElement;

— Запуск заполнения каждой родительской по отношению к текущей таблицы в отдельном потоке с передачей в метод fillParentTable() созданной выборки значений первичных ключей;

— Заполнение текущей таблицы с помощью генераторов столбцов с сопоставлением каждому из переданных значений первичного ключа дочерней таблицы нескольких сгенерированных записей;

— Запуск заполнения каждой дочерней таблицы (кроме той, из которой было запущено заполнение текущей) в отдельном потоке с передачей в метод fillParentTable() созданной выборки

Диаграмма последовательностей, описывающая алгоритм заполнения родительской таблицы, приведена на рисунке 16.

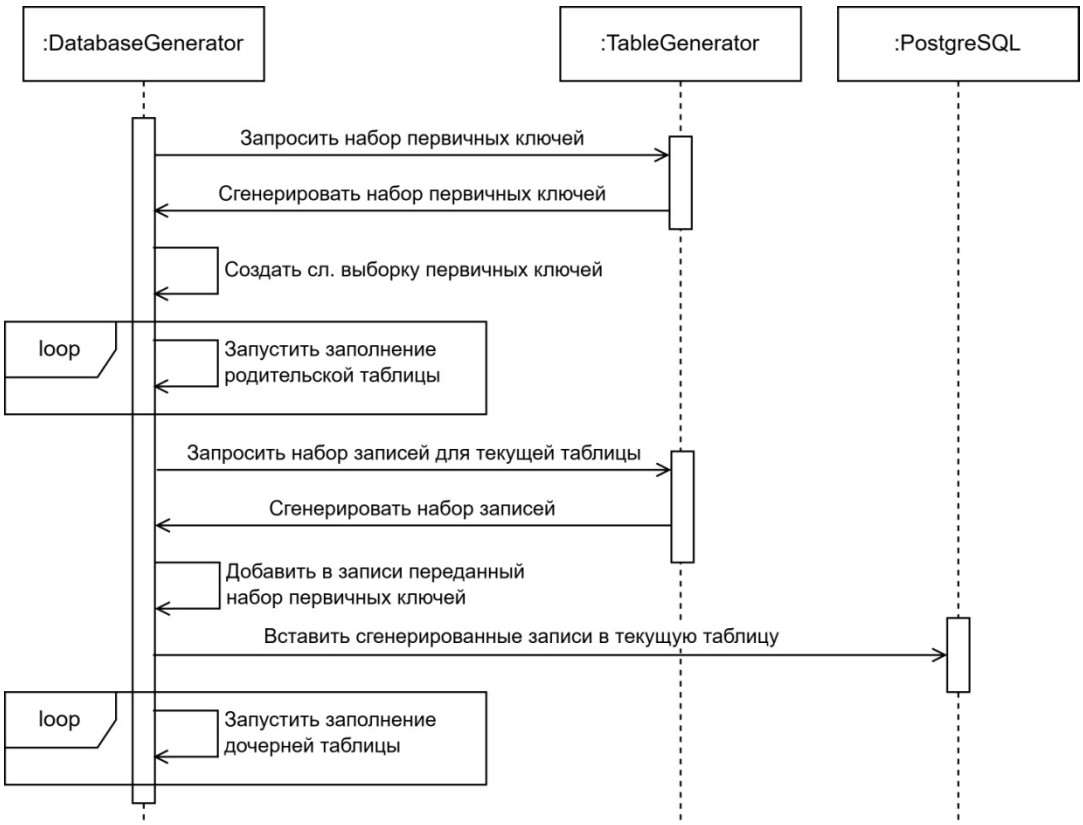

Рисунок 16 – Диаграмма последовательностей для метода fillParentTable()

Алгоритм заполнения дочерней таблицы отличается от генерации родительской таблицы тем, что вместо сопоставления каждому первичному ключу из переданного набора нескольких записей текущей таблицы, одной записи в текущей таблице сопоставляется несколько значений первичного ключа из набора (количество также определяется случайным распределением). При этом при создании данной записи производится обновление соответствующих записей в родительской таблице (где значение первичного ключа входит в вышеупомянутые несколько значений). Диаграмма последовательностей, описывающая алгоритм заполнения родительской таблицы, приведена на рисунке 17.

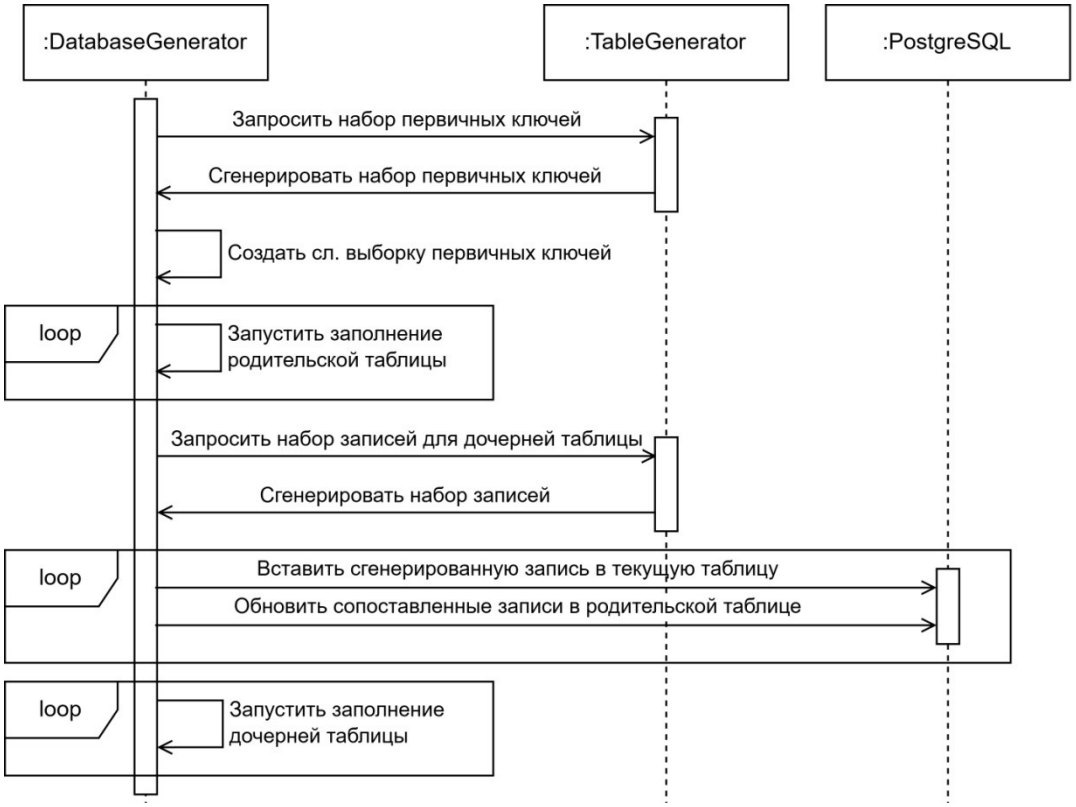

Рисунок 17 – Диаграмма последовательностей для метода fillChildTable()

3.5 Добавление ограничений

Последний шаг алгоритма работы приложеия – добавление ограничений в таблицы. Оно заключается в прохождении по всем ограничениям, записанным в экземпляре классе ConstraintSet, полученном на первом шаге алгоритма. Для каждого ограничения первичного ключа, внешнего ключа и уникальности исполняется соответствующий запрос к базе данных. Кроме этого, осуществляется прохождение по всем правилам, записанным в экземпляре класса RuleSet, и исполнение запросов на наложение ограничения обязательности на те столбцы, где в соответствующих правилах значение поля nullChance равно 0.

3.6 Поддержка многопоточности

Реализация многопоточности в системе производилась с помощью класса ThreadPoolExecutor из пакета java.util.concurrent стандартной библиотеки Java[8]. Алгоритм заполнения любой таблицы устроен таким образом, что запуск заполнения каждой родительской по отношению к ней таблицы производится одновременно, при чём каждая родительская таблица заполняется в отдельном потоке. Запуск заполнения каждой таблицы, дочерней по отношению к текущей, также производится одновременно.

Однако между запуском заполнения родительских таблиц и запуском заполнения дочерних таблиц производится заполнение текущей таблицы, что является долгой операцией, т.к. включает в себя обращение к базе данных. Следовательно, заполнение дочерних по отношению к текущей таблиц начинается значительно позже, чем заполнение родительских таблиц.

Кроме того, алгоритм заполнения дочерней таблицы включает в себя исполнение большего количества запросов к базе данных: происходит не только добавление записей в текущую таблицу, но и обновление записей в одной из родительских.

Из вышесказанного следует, что максимальная оптимизация времени работы программы возможна, если среди всех процедур заполнения таблиц минимальное количество из них будет являться процедурой заполнения дочерней таблицы.

4 Тестирование

Тестирование системы производилось на устройстве со следующими характеристиками:

— Центральный процессор Intel Core i7-8700: 6 ядер, частота 3.2 ГГц;

— Оперативная память типа DDR4: частота 2.4 МГц, объём 32 Гб;

— SSD Kingston sa400s37240g: скорость записи до 350 Мб/с.

При тестировании системы использовалась база, состоящая из 6 таблиц, ER-диаграмма которой приведена на рисунке 18.

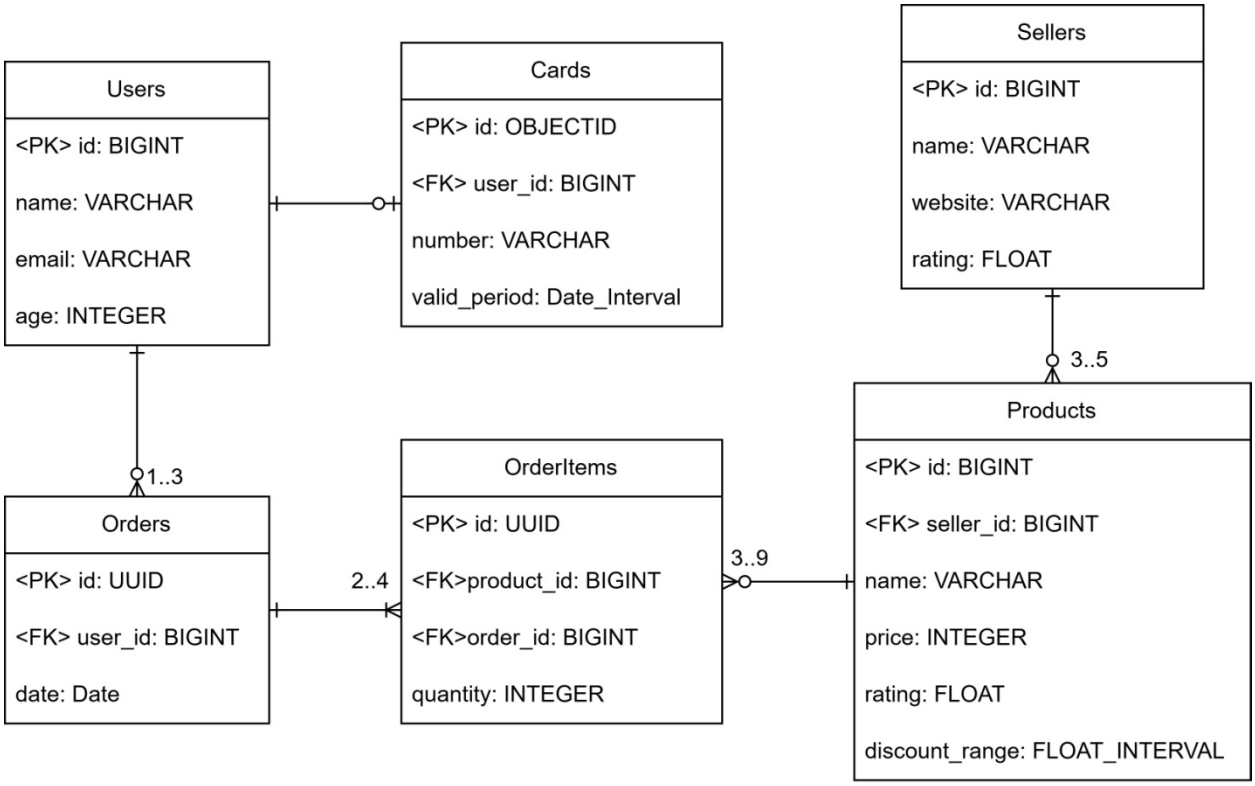

Рисунок 18 – ER-диаграмма тестовой базы данных

4.1 Влияние выбора начальной таблицы на производительность

Производительность системы зависит от того, с какой таблицы начинается процедура заполнения базы данных. Лучшие показатели достигаются при минимальном количестве вызовов процедуры заполнения

дочерней таблицы. В случае, когда стартовой является таблица Users, будет производиться всего 2 процедуры заполнения дочерней таблицы, а в случае, когда стартовой является таблица OrderItems, будет производиться 4 таких процедуры. Следовательно, заполнение базы данных в первом случае должно занять меньшее количество времени. Тестирование будет производиться для количеств записей в таблице Users, равным 1000, 2000, 3000, …, 9000 (во втором случае количество записей в начальной таблице OrderItems рассчитывается исходя из того, что на одну запись в таблице Users в среднем приходится 5.7 записей в таблице OrderItems). Результирующий график изображён на рисунке 19. Красной линией обозначен случай, где начальной является таблица Users, синей – случай, где начальной является таблица .

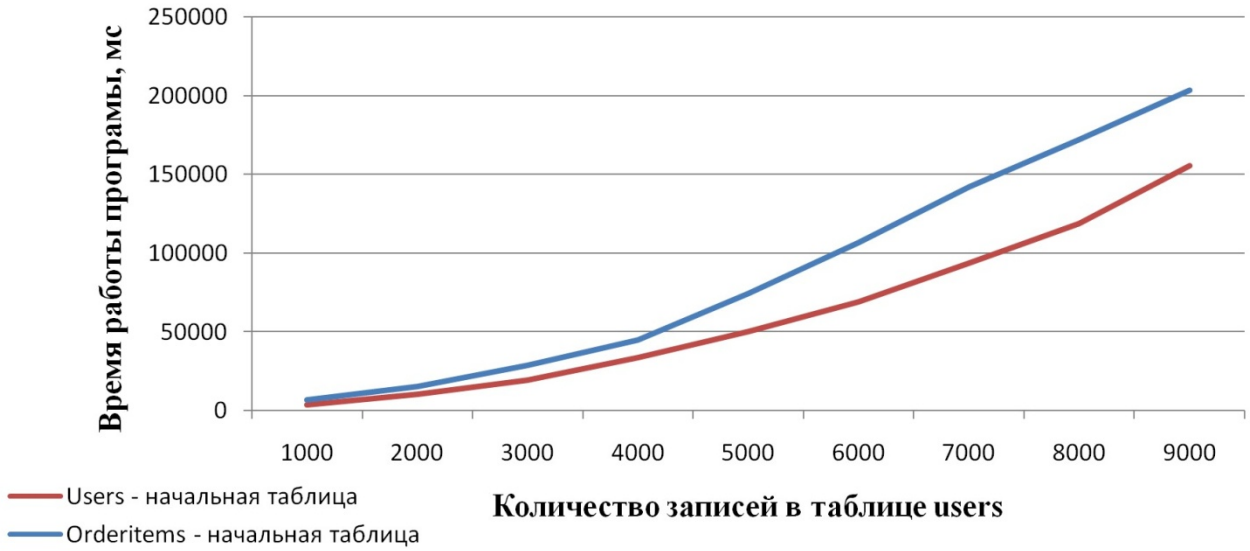

Рисунок 19 – Зависимость производительности от начальной таблицы

График показывает, что предположение, выдвинутое выше, верно, при этом разница значений времени работы в данном случае может достигать 25%.

4.2 Влияние многопоточности на производительность

Благодаря использованию многопоточности, в случае, когда начальной в заполнении базы данных является таблица Users, производительность системы достигает лучших показателей. При отсутствии применения многопоточности данный эффект не может быть достигнут. График, изображенный на рисунке 20, показывает это. Красной линией обозначен случай, когда многопоточность не используется, синей – случай, когда многопоточность используется.

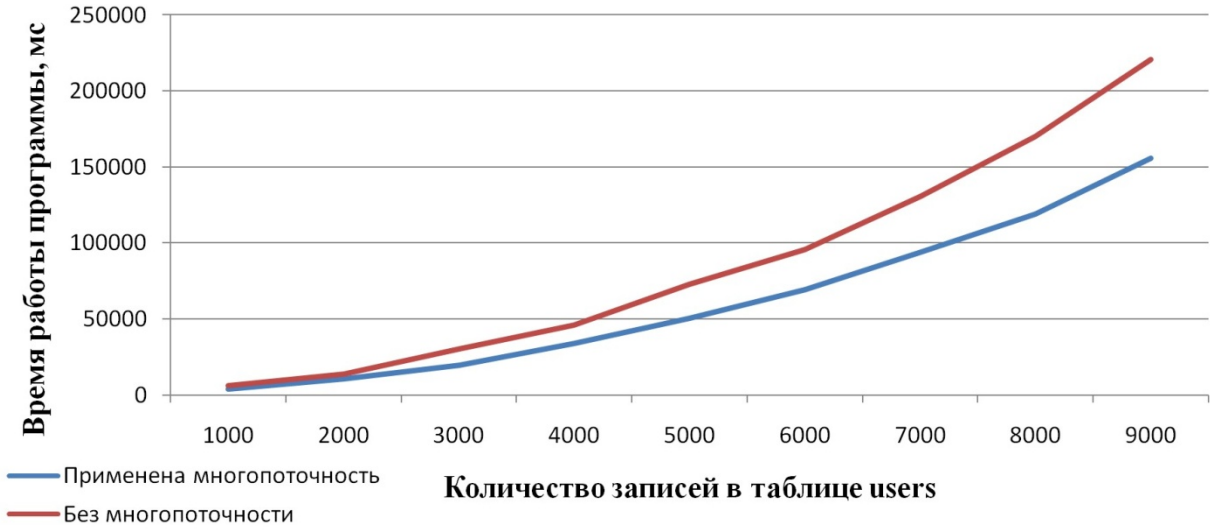

Рисунок 20 – Зависимость производительности от многопоточности

График показывает, разница во времени работы программы в однопоточном и многопоточном режимах может достигать 30%.

ЗАКЛЮЧЕНИЕ

В ходе выполнения курсовой работы было реализовано серверное приложение для генерации анонимизированных баз данных, позволяющее формировать тестовые данные, соответствующие заданной структуре с учётом основных типов данных и ограничений целостности.

Приложение обеспечивает возможность работы со следующими типами данных:

— Целые числа;

— Числа с плавающей точкой;

— Строки;

— Даты;

— Числовые интервалы;

— Интервалы дат.

Приложение учитывает следующие ограничения целостности данных:

— Первичный ключ;

— Внешний ключ;

— Уникальный ключ;

— Обязательные столбцы.

Приложение обеспечивает возможность использования в качестве типа данных первичного ключа следующих типов данных:

— Целое число;

— UUID;

— ObjectID.

Приложение поддерживает возможность работы с двумя типами кардинальности связей между реляционными таблицами, а именно «Один к одному» и «Один ко многим». Также обеспечена возможность задания правила того, является ли обязательным наличие сущности (сущностей) с

левой и с правой стороны связи. В случае связи «Один ко многим» приложение позволяет задать правило, согласно которому определяется количество сущностей с правой стороны связи.

Приложение поддерживает возможность многопоточного заполнения базы данных, т.е. одновременного (параллельного) заполнения нескольких таблиц.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Российская Федерация. Законы. О персональных данных : Федеральный закон № 266-ФЗ : [принят Государственной думой 16 сентября 2003 года : одобрен Советом Федерации 24 сентября 2003 года]. – (Дата обращения 05.05.2025).

2. О защите физических лиц при обработке персональных данных и о свободном обращении таких данных. Регламент № 2016/679 Европейского парламента и Совета Европейского Союза [принят 27 апреля 2016 года] . – (Дата обращения 05.05.2025).

3. Васильев, А. Н. Java для всех / А. Н. Васильев. – Санкт-Петербург : Изд-во Питер, 2020. – 512 с. – Текст: Непосредственный (Дата обращения 10.05.2025).

4. Рогов, Е. В. PostgreSQL изнутри / Е. В. Рогов. – Москва : Изд-во ДМК Пресс, 2023. – 663 с. – Текст: Непосредственный (Дата обращения 15.05.2025).

5. Allen, J. F. Maintaining knowledge about temporal intervals – Текст: Непосредственный //Communications of the ACM. – 1983. – Vol. 26, № 11. – С.832 - 843 (Дата обращения 15.05.2025).

6. Представление дат и времени. Международный стандарт ISO-8601 [Опубликован 1 февраля 2019 года Международной организацией по стандартизации (ISO)]. – (Дата обращения 12.05.2025).

7. Гмурман, В.Е. Теория вероятностей и математическая статистика : учебное пособие / В. Е. Гмурман. – 9-е издание. – Москва : «Высшая школа», 2004. – 407 с. – Текст: Непосредственный (Дата обращения 11.05.2025).

8. Java Concurrency на практике / Б. Гетц, Т. Пайерлс, Д. Блох [и др.]; – Санкт-Петербург : Изд-во Питер, 2025. – 464 с. – Текст: Непосредственный (Дата обращения 15.05.2025).