МИНОБРНАУКИ РОССИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
“ВОРОНЕЖСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ”
(ФГБОУ ВО «ВГУ»)

Факультет компьютерных наук

Кафедра программирования и информационных технологий

Разработка генератора анонимизированных баз данных

Курсовая работа по дисциплине

?

6 семестр 2024/2025 учебного года

09.03.04 Программная инженерия

Зав. кафедрой _____ к.т.н., доцент С.Д.Махортов

Обучающийся _____ ст. 3 курса М.А.Ячный

Руководитель _____ ст.преп. Н.К.Самойлов

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 консольного 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;

— Уникальность.

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, задаётся в отдельных файлах формата JSON.

2 Анализ предметной области

2.1 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.

Маскировка также позволяет учитывать ограничения целостности данных. Например, при работе с первичными ключами можно генерировать уникальные значения, а для внешних ключей — создавать согласованные наборы данных, сохраняющие связи между таблицами. Это делает тестовые среды максимально приближенными к реальным условиям эксплуатации системы.

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

2.2 Способы генерации чисел

В контексте генерации тестовых данных принципиальное значение имеет корректная имитация числовых значений, приближенных к реальным данным. Важно отметить, что в практических сценариях числовые данные редко являются "по-настоящему случайными" в строгом математическом смысле. Напротив, они подчиняются определенным вероятностным распределениям, отражающим закономерности предметной области.

В теории вероятностей и математической статистике все случайные величины принципиально разделяются на два класса: дискретные и непрерывные. Это фундаментальное различие определяет не только их математические свойства, но и методы генерации соответствующих значений при создании тестовых данных.

Дискретные случайные величины принимают отдельные, изолированные значения (в контексте разрабатываемого приложения – целочисленные), в то время как непрерывные случайные величины могут принимать любое значение в некотором интервале. В разрабатываемом

приложении случайной величиной такого типа будут являться числа с плавающей точкой.

При генерации целочисленных значений (дискретных данных) наиболее часто используются три распределения: равномерное, биномиальное и распределение Пуассона.

Равномерное распределение характеризуется одинаковой вероятностью появления любого значения из заданного диапазона. Применяется в случаях, когда все исходы равновероятны.

Биномиальное распределение описывает количество успехов в серии независимых испытаний с постоянной вероятностью успеха.

Распределение Пуассона применяется для моделирования редких событий в фиксированный промежуток времени. Типичные примеры использования включают генерацию количества запросов к серверу в минуту или число ошибок на странице текста.

Для генерации вещественных чисел (непрерывных величин) наибольшее практическое значение имеют три распределения: равномерное, нормальное и экспоненциальное.

Равномерное непрерывное распределение аналогично дискретному случаю, все значения из заданного интервала имеют равную плотность вероятности. Используется при моделировании величин без выраженных закономерностей.

Нормальное распределение – крайне важное для статистики распределение, описывающее множество природных и социальных явлений. Применяется для генерации данных, сосредоточенных вокруг среднего значения с постепенным уменьшением частоты отклонений, таких как рост людей или погрешности измерений.

Экспоненциальное распределение описывает временные интервалы между независимыми событиями, происходящими с постоянной средней интенсивностью. Широко используется при моделировании времени

обслуживания запросов, длительности сеансов или сроков эксплуатации оборудования.

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. Разработанная Джеймсом Алленом в 1983 году, эта алгебра находит широкое применение в планировании задач, анализе временных рядов и, что особенно важно для нашей задачи, в генерации тестовых данных. 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: начало A до начала B, конец A после начала B, но до конца B;
— Обратное перекрытие: зеркальное отношение к перекрытию;
— Следование: конец A точно совпадает с началом B;
— Обратное следование: зеркальное отношение к непосредственному следованию;
— Включение: A полностью содержится внутри B;
— Обратное включение: B полностью содержится внутри A;

3 Практическая часть

Для реализации генератора анонимизированных баз данных был выбран язык Java в сочетании с системой управления базами данных PostgreSQL. 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.

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

PostgreSQL была выбрана в качестве целевой СУБД благодаря ее развитой системе типов данных и мощной поддержке ограничений целостности. Эта система предлагает широкие возможности для определения сложных структур данных и обеспечения их согласованности через механизмы первичных (PRIMARY KEY) и внешних ключей (FOREIGN KEY), а также ограничений уникальности (UNIQUE).

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 и создание специализированных генераторов значений для каждой колонки таблиц на основе правил, заданных в конфигурационном файле. На этом этапе программа анализирует типы данных, требуемые распределения, допустимые диапазоны значений, кратность связей между таблицами.

Второй этап заключается в непосредственной генерации данных. Используя созданные на предыдущем шаге генераторы, программа заполняет

таблицы синтетическими данными, соблюдая заданный объём записей, статистические закономерности и взаимную согласованность данных в связанных таблицах.

Третий этап посвящен применению ограничений целостности. Программа добавляет в сформированные таблицы ограничения первичных ключей (PRIMARY KEY), внешних ключей (FOREIGN KEY), ограничения уникальности (UNIQUE) и обязательные поля (NOT NULL).

Реализация каждого этапа будет подробно рассмотрена в следующих разделах с приведением диаграмм, описывающих структуру приложения и алгоритмы.

3.1 Исходные данные

Данный этап целиком посвящен анализу исходных данных и подготовке к непосредственной генерации анонимизированных данных. Как было описано в разделе 1, исходные данные и правила записаны в нескольких файлах формата json.

Первый из них должен описывать 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 1.

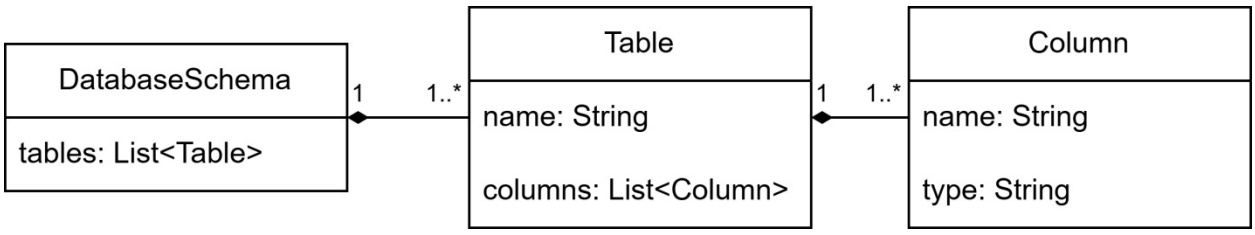

Рисунок 1 – Диаграмма классов схемы базы данных

Во втором файле формата JSON задаются правила, по которым будут генерироваться значения в столбцах, не являющихся первичными или внешними ключами. Все правила хранятся в одном экземпляре корневого класса RuleSet. В данном классе для каждого из допустимых в приложении типа данных содержится список правил, касающихся правил генерации значений в столбцах этого типа данных.

Для каждого типа данных правила генерации значений этого типа описаны собственным классом, однако все классы правил реализуют общий интерфейс Rule, имеющий 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 шанса, 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 2 представлена диаграмма классов для набора правил генерации значений. Классы для правил конкретных типов будут подробно описаны далее в этом разделе.

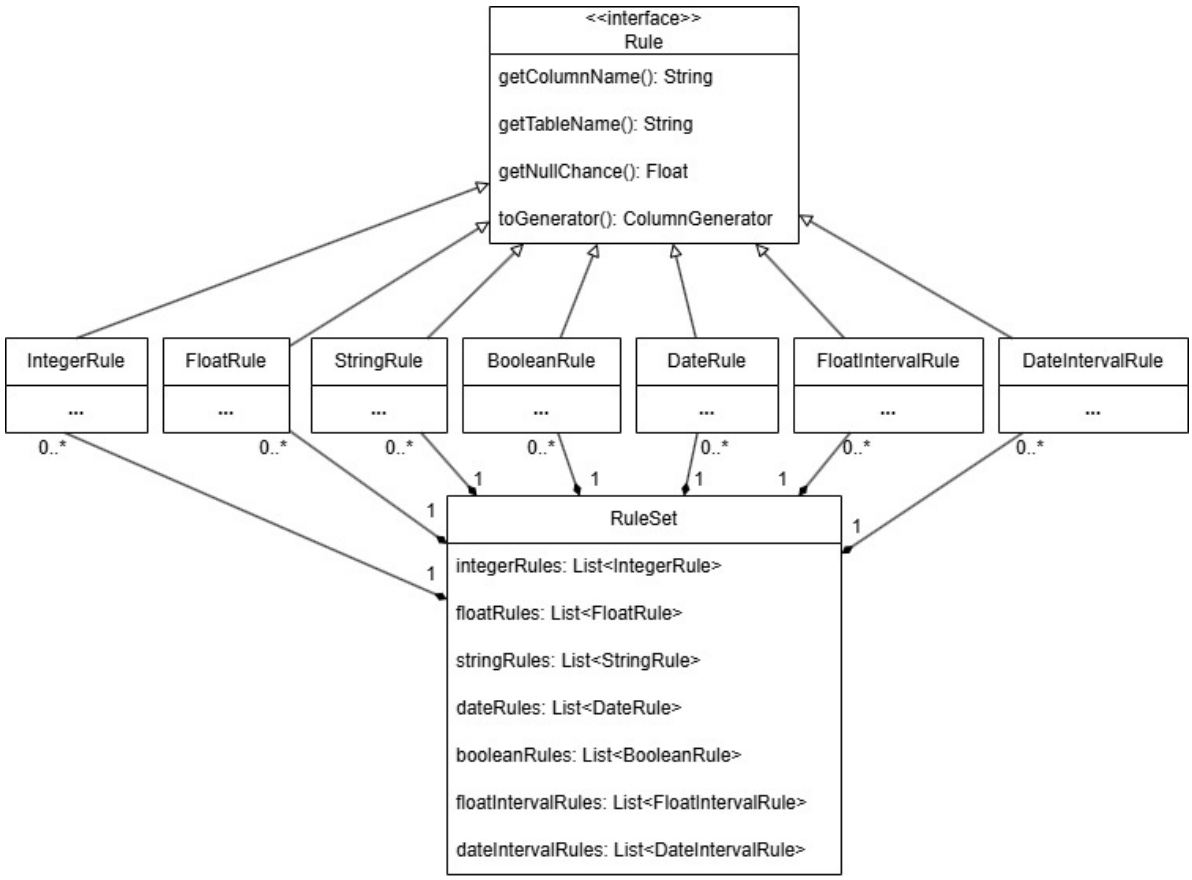

Рисунок 2 – Диаграмма классов набора правил генерации

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 3 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.

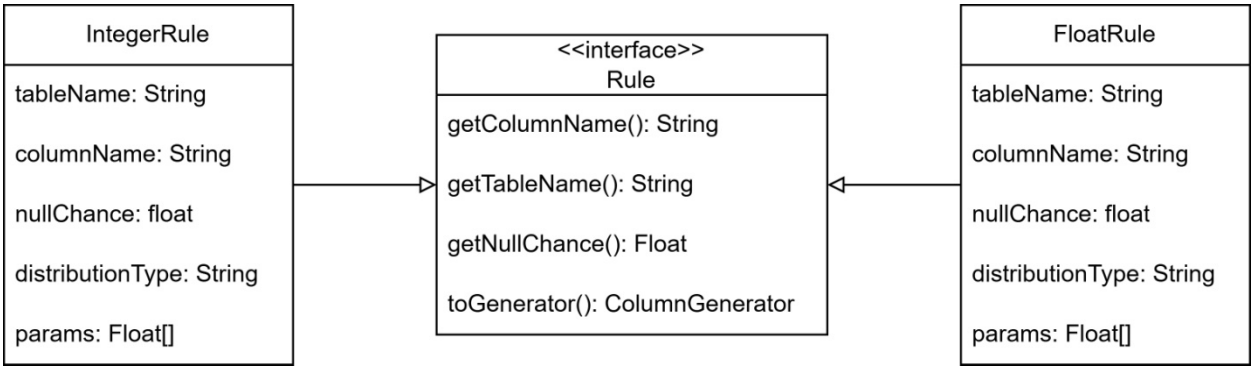

Рисунок 3 – Диаграмма классов для правил целых численных значений

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 4 приведена диаграмма классов для правила генерации строковых значений.

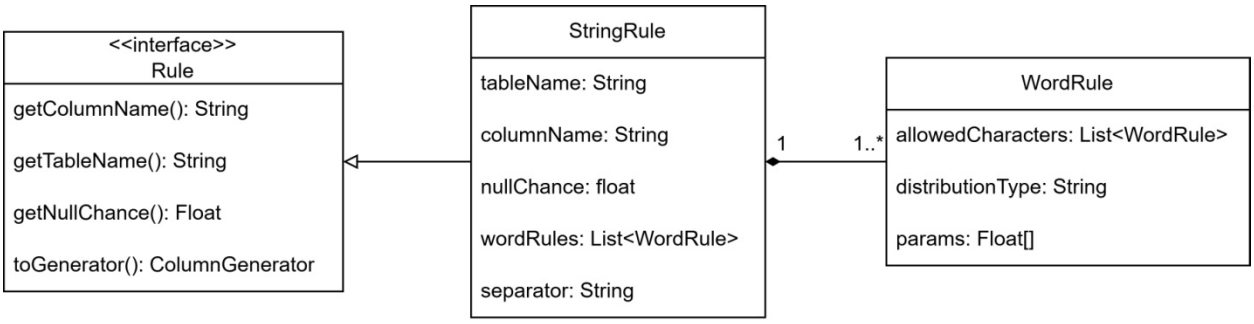

Рисунок 4 – Диаграмма классов для правил генерации строковых значений

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[3]. На рисунке 5 представлена диаграмма классов для данного вида правила.

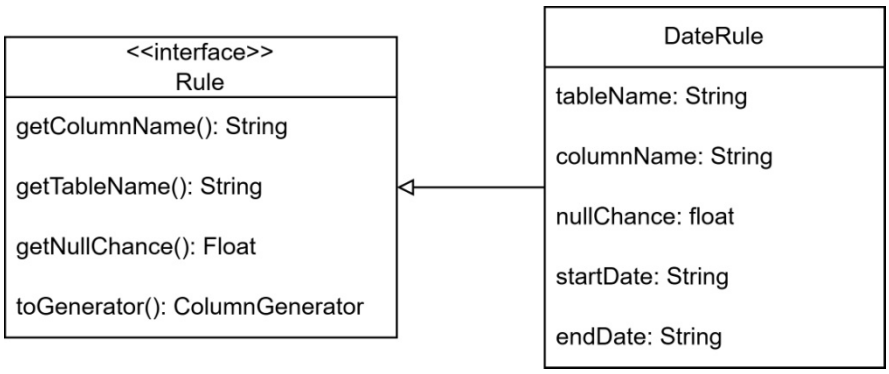

Рисунок 5 – Диаграмма классов для правил генерации дат

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 6.

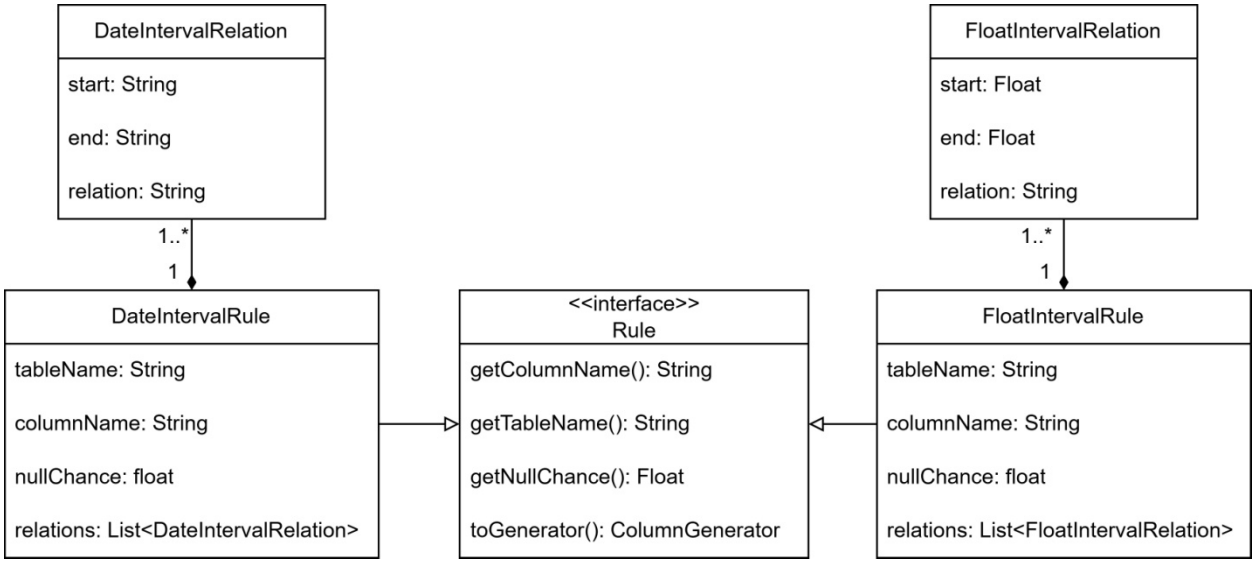

Рисунок 6 – Диаграмма классов для правил генерации интервалов

Третий файл формата JSON 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 7.

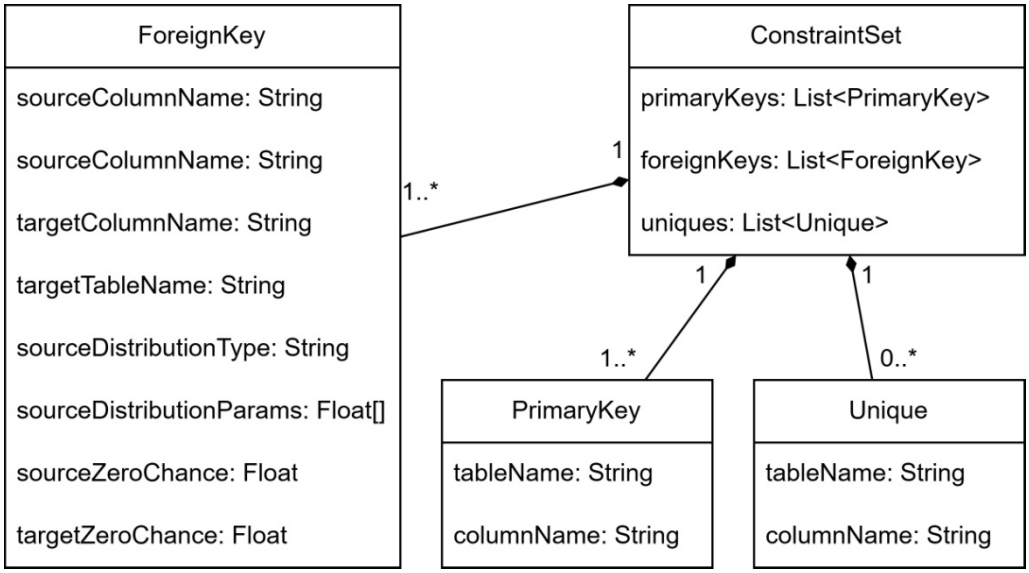

Рисунок 7 – Диаграмма классов для набора ограничений

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

3.2 Генерация значений разных типов

Вся 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 поле: 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() просто генерирует очередной набор значений без учёта ограничения уникальности. Используется именно массив строк, т.к. генератор может отвечать за заполнение сразу нескольких колонок (в данной работе примером являются генераторы интервалов). Метод getNextValues() имеет разные алгоритмы работы в зависимости от значения unique, более подробно они описаны на рисунке 8.

```
if (unique):
    do{
        values = generateValues();
    } while (ещё не были сгенерированы
            значения, равные values);
    return values;
else:
    return generateValues();
```

Рисунок 8 – Алгоритм метода getNextValues()

Интерфейс PrimaryKeyGenerator устроен значительно проще и имеет всего 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 колонки и тип данных первичного ключа. Диаграмма классов для генератора базы данных приведена на рисунке 9. Необходимо отметить, что для класса DatabaseGenerator в данной диаграмме приведено неполное описание. Оставшиеся поля и методы класса будут описаны позже в данном разделе.

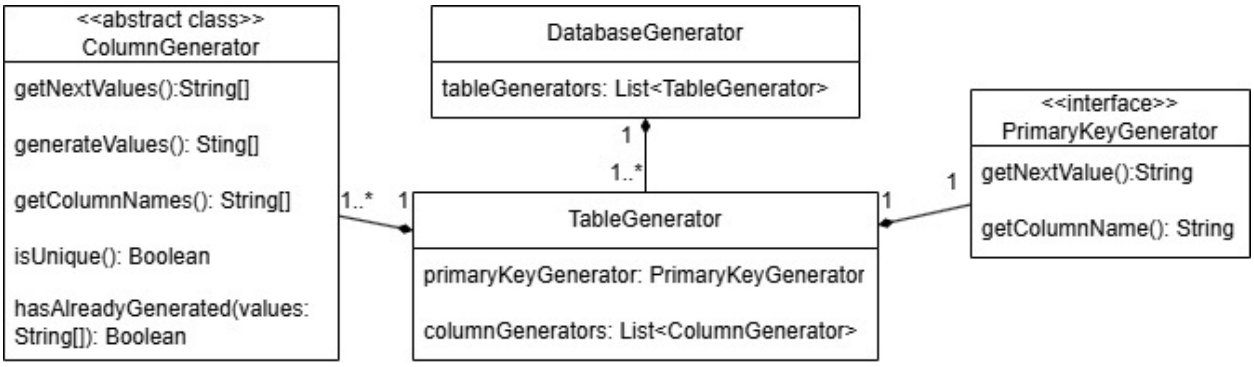

Рисунок 9 – Диаграмма классов для генераторов таблиц

Кроме создания генераторов столбцов, перед началом 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 колонки 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 10.

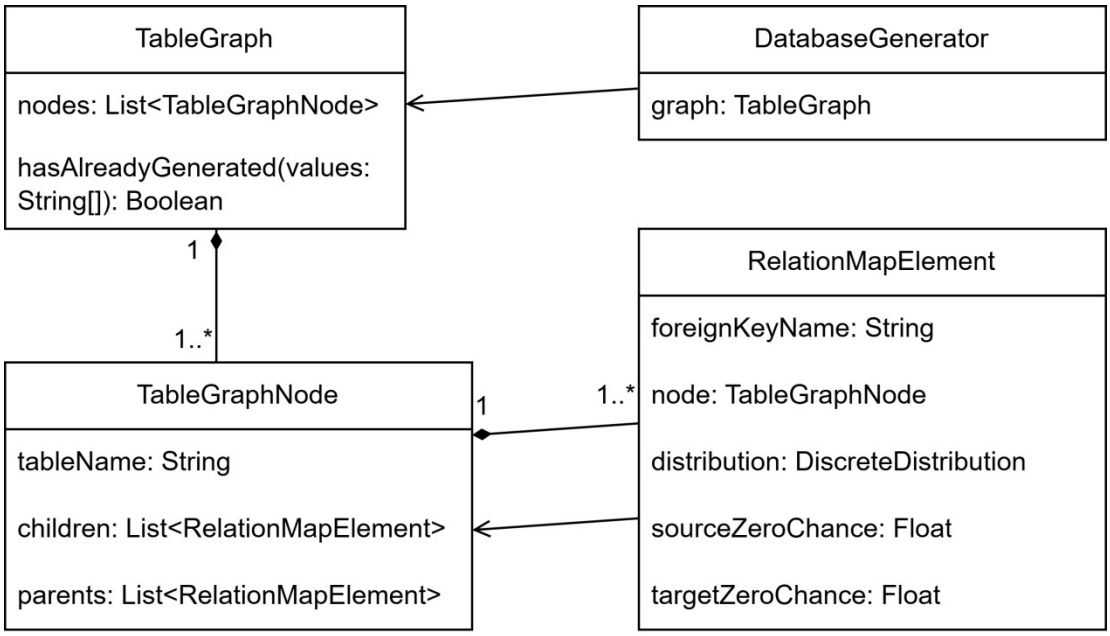

Рисунок 10 – Диаграмма классов для графа отношения таблиц

За генерацию численных значений отвечают реализующие австрактный класс ColumnGenerator классы IntegerGenerator и FloatGenerator. Данные два класса отличаются только полем distribution, которое в случае IntegerGenerator имеет тип DiscreteDistribution, а в случае FloatGenerator – ContinuousDistribution.

Генерация значений для равномерного дискретного распределение, равномерного непрерывного распределения, а также нормального распределения осуществляется крайне просто благодаря встроенным в класс Random из стандартного пакета java.util методам nextInt(), nextFloat() и nextGaussian(). Однако для биномиального распределения, экспоненциального распределения и распределения Пуассона необходимо использовать собственные алгоритмы.

Самым простым способом генерации значений, подчинённых биномиальному распределению, является проведение эксперимента Бернулли, которое описывается данным типом распределения. Алгоритм генерации значений данного типа описан на рисунке 11. Здесь n – количество проведённых испытаний, p – вероятность успеха в отдельном испытании. randInt(m, k) – функция, возвращающая случайное число с плавающей точкой от m до k.

```
result = 0
for(i от 0 до n):
    if(rand(0,1) < p):
        result++
return result
```

Рисунок 11 – Алгоритм генерации значений биномиального распределения

Для генерации значений, подчинённых экспоненциальному распределению, будет применён метод с использованием обратной функции, позволяющий связать значение, подчинённое равномерному распределению,

с экспоненциальным распределением[4]. Вычисление производится по следующей формуле, где u – случайное число с плавающей точкой от 0 до 1:

$$x = -\ln(1 - u)$$

Генерация значений, подчинённых распределению Пуассона, также будет производиться с помощью алгоритма, использующего равномерные случайные величины[5]. На рисунке 12 изображён алгоритм генерации значений, подчинённых распределению Пуассона. Здесь λ – параметр случайного распределения.

```
N = 0
P = 1
while(P > exp(-λ)):
    u = rand(0, 1)
    N = N + 1
    P = P * u
return N
```

Рисунок 12 – Алгоритм генерации значений распределения Пуассона

Как было описано в подразделе 3.1, генерируемая строка представляет собой набор сгенерированных слов, соединённых заданной последовательностью символов. Класс StringGenerator имеет поле separator, задающее данную последовательность, а также список wordGenerators, каждый элемент которого – экземпляр класса WordGenerator, отвечающий за генерацию отдельного слова.

Класс WordGenerator содержит поле allowedCharacters типа List<String>, содержащее наборы разрешённых к использованию в слове символов. Вторым полем класса является дискретное распределение lengthDistribution. Алгоритм генерации строки зависит от того, что находится в поле lengthDistribution. Если там пустое значение, то считается, что длина слова

постоянная и равна длине списка allowedCharacters, а каждый его элемент задаёт набор разрешённых символов для конкретной позиции в слове. В обратном случае длина слова определяется следующим сгенерированным распределением целым числом, а allowedCharacters содержит единственный элемент, задающий единый для всех позиций в слове набор разрешённых символов. Алгоритм генерации строкового значения приведён на рисунке 13.

```
words = new List();
for(для каждого wordGenerator в WordGenerators):

    word = [пустая_строка]
    if(wordGenerator.lengthDistribution = null):
        for(i от 0 до длины allowedCharacters):
            word += случ_символ(allowedCharacters[i])
    else:
        for(i от 0 до lengthDistribution.nextInt()):
            word += случ_символ(allowedCharacters[0])
    words.add(word)

return join(words, separator)
```

Рисунок 13 – Алгоритм генерации строки

Генерация интервалов сводится к заданию 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 задаёт свой набор из данных четырёх значений. При этом, как было обозначено в подразделе 3.1, все отношения должны выполняться одновременно, следовательно, для значений, задающих верхние границы, выбираются минимальные значения из всех отношений, а для нижних границ – максимальные.

3.3 Заполнение таблиц

Заполнение базы данных начинается с любой из указанных в исходных данных таблиц. Для заполнения первой таблицы в классе DatabaseGenerator имеется метод fillFirstTable(), принимающий на вход генератор данной таблицы, соответствующий ей узел графа отношений таблиц graph, и количество записей, которые необходимо сгенерировать.

Алгоритм генерации первой таблицы начинается с генерации набора первичных ключей соответствующего типа. После этого сразу запускается заполнение всех родительских таблиц с передачей в метод fillParentTable() сгенерированного набора первичных ключей (таблиц, первичные ключи которых указывают на текущую таблицу), т.к. для заполнения родительской таблицы необходим только набор первичных ключей дочерней таблицы, из которой запускается заполнение. Значения остальных столбцов значения не имеют, поэтому запуск заполнения родительских таблиц может производиться до непосредственной вставки записей в соответствующую дочернюю таблицу. Необходимо отметить, что в метод fillParentTable() передаётся не весь сгенерированный набор ключей, а случайная выборка из него, объём которой задаётся значением поля sourceZeroChance соответствующего экземпляра класса relationMapElement (если значение равно 0, то передаётся весь набор). Заполнение каждой родительской таблицы производится в отдельном потоке.

После запуска заполнения родительских таблиц производится непосредственное заполнение первой таблицы с помощью генераторов столбцов, содержащихся в переданном генераторе таблицы, и сгенерированном на предыдущем шаге наборе первичных ключей.

Начинать заполнение дочерних таблиц можно лишь после этого шага, т.к. алгоритм заполнения дочерней таблицы включает в себя изменение существующих записей соответствующей родительской таблицы. Заполнение каждой дочерней таблицы также производится в отдельном потоке и в метод

fillChildTable также передаётся не весь набор первичных ключей, а случайная выборка, объём которой определяется значение поля targetZeroChance соответствующего экземпляра класса relationMapElement (если значение равно 0, то передаётся весь набор).

Алгоритм заполнения родительской таблицы также начинается с генерации набора первичных ключей и запуска заполнения указывающих на текущую таблицу родительских таблиц с передачей в метод fillParentTable() случайной выборки из набора первичных ключей. В отличие от метода заполнения первой таблицы, при непосредственной вставке записей в таблицу кроме набора первичных ключей и значений, генерируемых генераторами столбцов, используется переданный из дочерней таблицы набор первичных ключей. Каждому значению из данного набора ставится в соответствие несколько записей в текущей таблице, количество которых определяется случайным распределением, хранящимся в поле distribution соответствующего экземпляра класса relationMapTable. В каждой из данных записей столбец соответствующего внешнего ключа будет заполнен значением первичного ключа из переданного набора. Кроме этого, в текущей таблице создаётся несколько записей, где в столбец внешнего ключа будет помещено пустое значение. Количество данных записей задаётся значением поля targetZeroChance соответствующего экземпляра класса relationMapElement. Далее таким же образом производится запуск заполнения дочерних таблиц.

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). Также в

текущей таблице производится создание нескольких записей, которым не соответствуют никакие значения первичных ключей в родительской таблице. Количество таких записей определяется значением поля sourceZeroChance соответствующего экземпляра класса relationMapElement. Затем, аналогичным предыдущим двух методам образом, производится запуск дочерних таблиц.

3.4 Добавление ограничений

Последний шаг алгоритма заполнения базы данных – 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.

3.5 Измерение производительности

Как было описано в разделе 3.3, алгоритмы заполнения родительских и дочерних таблиц имеют различия, последний включает в себя исполнение большего количества запросов к базе данных. Кроме того, при заполнении любой текущей таблицы запуск заполнения дочерних по отношений к ней таблиц осуществляется значительно позже, чем запуск заполнения родительских таблиц. Следовательно, оптимизировать время работы программы возможно, если среди всех процедур заполнения таблиц минимальное количество из них будет являться процедурой заполнения дочерней таблицы. Это можно достичь посредством правильного подбора

одного из входных параметров – имени таблицы, с которой начнётся заполнение базы данных.

Тестирование производительности будет производиться на примере базы данных, описывающей некий маркетплейс и состоящей из 6 таблиц. ER-диаграмма для базы данных приведена на рисунке 14.

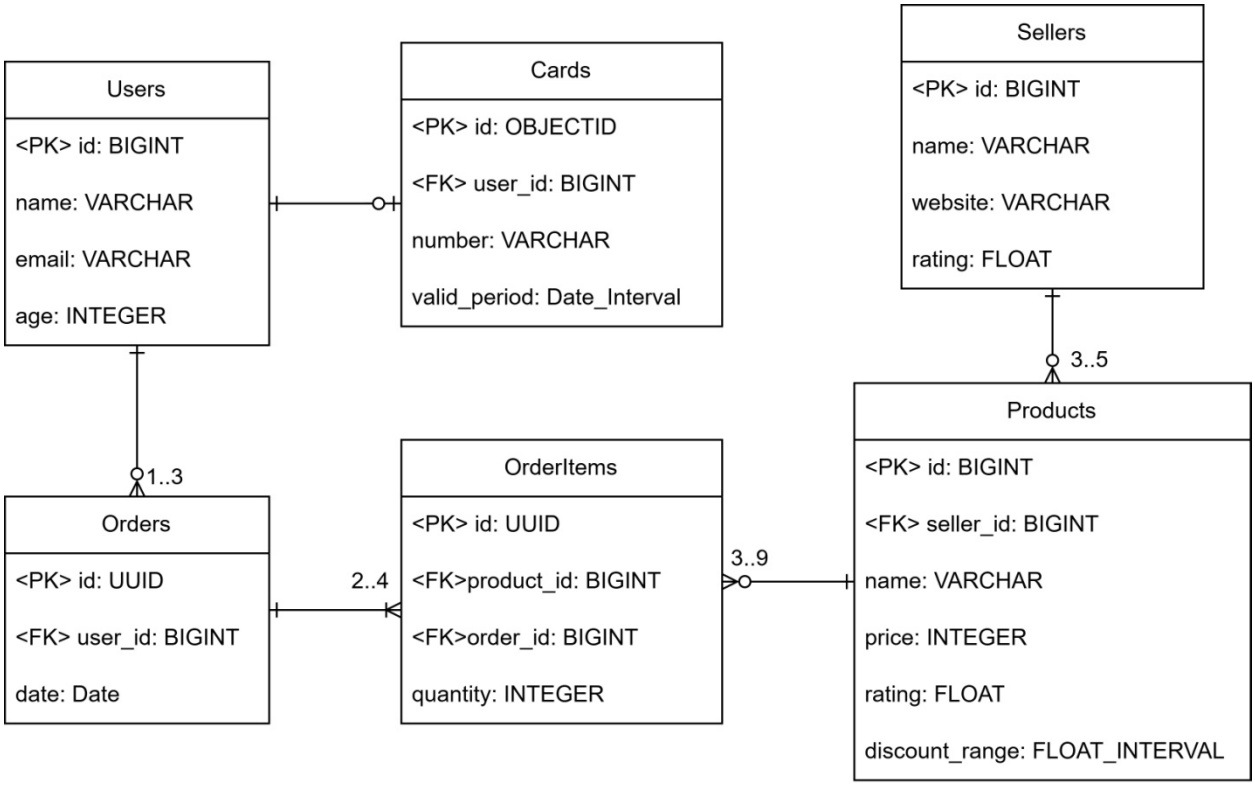

Рисунок 14 – ER-диаграмма тестовой базы данных

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 15. Красной линией представлен первый случай заполнения базы данных, синей – второй.

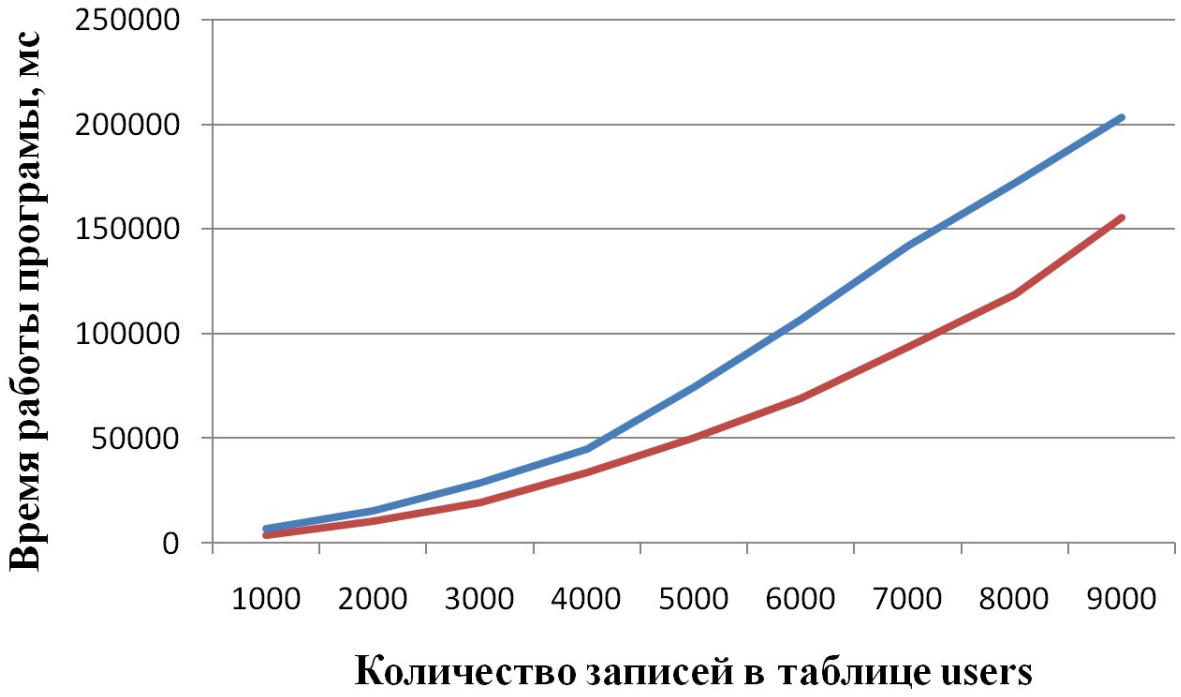

Рисунок 15 – График измерения производительности

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

ЗАКЛЮЧЕНИЕ

В ходе выполнения курсовой работы была успешно разработана система генерации анонимизированных тестовых баз данных, представляющая собой комплексное решение для обеспечения безопасной работы с информацией в процессах разработки и тестирования программного обеспечения. Реализованный генератор сочетает в себе гибкость настройки, высокую производительность и строгое соблюдение принципов защиты персональных данных.

Особую ценность разработанное решение приобретает в контексте современных требований к защите персональных данных. Генератор не только устраняет риски, связанные с использованием реальной информации в тестовых средах, но и обеспечивает соответствие нормативным требованиям таких стандартов как GDPR и ФЗ-152 "О персональных данных".

Перспективы развития проекта включают расширение поддерживаемых СУБД, добавление новых алгоритмов анонимизации, а также создание графического интерфейса для упрощения работы с системой. Особый интерес представляет интеграция возможностей машинного обучения для генерации более реалистичных синтетических данных, сохраняющих все статистические закономерности исходных наборов.

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 13.05.2025).

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. Представление дат и времени. Международный стандарт ISO-8601 [Опубликован 1 февраля 2019 года Международной организацией по стандартизации (ISO)]. – (Дата обращения 10.05.2025).

4. Generating Random Variables From A Uniform Distribution [Электронный ресурс] – URL:https://telecombook.net/c1_s6b_generation_RV_from_Uniform.html – (Дата обращения 13.05.2025).

5. Simulating Poisson random variables – Direct method [Электронный ресурс] – URL:https://hpaulkeeler.com/simulating-poisson-random-variables-direct-method/ – (Дата обращения 13.05.2025).