Разработка генератора тестовых данных

Ж. С. Данилова, email: danilovazhanna9@gmail.com
М. В. Матвеева, email: marie.matveeva@gmail.com

Воронежский государственный университет

Аннотация. *В данной работе рассматривается процесс проектирования веб-приложения для генерации тестовых данных для баз данных Oracle. В статье описана функциональность приложения, основные требования, по которым была построена модель данных и спроектирован интерфейс приложения для генерации тестовых данных.*

Ключевые слова: *тестовые данные, Oracle, генерация данных, базы данных.*

Введение

Процесс тестирования ведется в течение всего времени разработки программного обеспечения. Для проверки работоспособности разработанных программных компонентов системы очень часто требуются тестовые денные. Данные не обязательно должны быть реальными, но они должны удовлетворять требованиям структуры, ограничениям и определенным бизнес-правилам.

Вследствие этого, с целью упрощения тестирования были созданы генераторы тестовых данных. Они позволяют генерировать случайные данные по заданным критериям, что значительно ускоряет процесс тестирования и разработки программного обеспечения.

Кроме этого, данные, полученные путем генерации, могут быть использованы и в обучающем процессе. Например, при обучении языку структурированных запросов SQL студенты могут практиковаться на сгенерированных данных.

1. Анализ существующих решений

Наиболее известные сервисы, позволяющие генерировать тестовые данные это: Databene Benerator, Mockaroo, Data Generation for SQL Server (Devart), Faker.

Результаты сравнения перечисленных сервисов представлены в таблице.

Таблица

Сравнительная таблица

Критерии сравнения	Databene Benerator	Mockaroo	Data Generation	Faker
Возможность создавать несколько связанных таблиц	+	-	+	-
Готовые наборы данных	+	+	+	+
Возможность создавать свои наборы	+	-	+	-
Web-приложение	-	+	-	-
Поддержка русского языка	-	-	-	-
Создание шаблона по готовому скрипту	+	-	-	-
Современный дизайн	-	+	+	-
Бесплатный доступ	+	+	-	+
Возможность авторизации для сохранения созданных схем	+	-	+	-

Проанализировав четыре различных сервиса, можно сделать вывод о том, что возможность создания нескольких таблиц поддерживают далеко не все сервисы. Также отсутствует хоть и не критичный, но все же желаемый пункт, как поддержка русского языка.

Кроме того, многие сервисы не поддерживают загрузку и распознавание скриптов с командами создания таблиц (CREATE TABLE), хотя это существенно упростило бы работу, так как создавать таблицы, атрибуты, ограничения с помощью пользовательского интерфейса приложения при большом объеме таблиц трудозатратно.

Также стоит отметить, что веб-приложение намного удобнее в использовании по сравнению с десктопным приложением, так как нет необходимости в установке дополнительного программного обеспечения.

Из рассмотренного следует необходимость создания приложения, которое будет учитывать недостатки существующих сервисов и поддерживать их преимущества.

2. Общая структура и функциональность приложения

Функциональные возможности web-приложения:

– чтение файлов с расширением sql, содержащих команды создания таблиц;

– синтаксический анализ скриптов создания таблиц;
– отображение структуры загруженных таблиц;
– настройка параметров генерации данных;
– генерация данных для нескольких таблиц;
– удаление, добавление, редактирование таблиц;
– генерация первичных ключей;
– генерация sql-команд вставки данных в таблицы;
– сохранение файла с результатом генерации в виде скриптов INSERT в формате sql;
– сохранение в профиле пользователя данных о настройках таблиц для повторной генерации.

3. Интерфейс приложения

Рассмотрим основную функциональность приложения на примере спроектированного интерфейса.

На рис. 1 представлена главная страница приложения, которую пользователь видит при запуске.

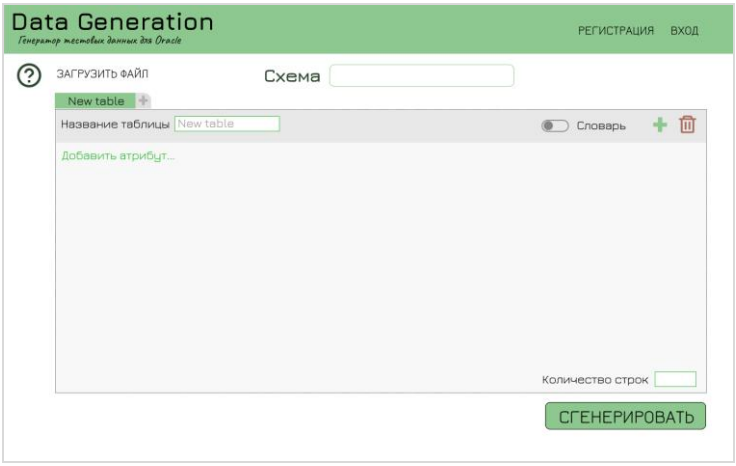

Рис. 1. Главная страница приложения

У неавторизованного пользователя есть возможность генерации данных, однако он не сможет сохранять созданные схемы и при каждом входе ему придется заполнять данные о таблицах и атрибутах на форме заново.

После этого пользователю предоставлен выбор: либо создавать вручную таблицы и атрибуты, либо загрузить скрипт.

Для того, чтобы вручную создавать таблицы пользователю требуется нажать на панели вкладок на значок «+». Затем появится диалоговое окно (рис. 2).

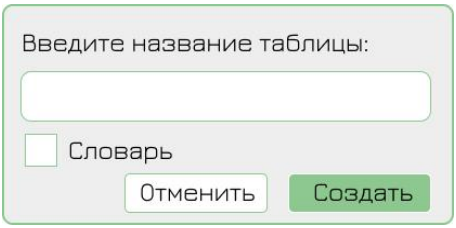

Рис. 2. Диалоговое окно создания таблиц

Существуют два разных типа таблиц: «Словарь» (или справочник) и «Сложная таблица». Это обусловлено тем, что для заполнения их данными требуются разные алгоритмы.

Для того, чтобы загрузить таблицы из файла нужно нажать на кнопку «Загрузить файл» и выбрать нужный файл со скриптами в своей файловой системе.

На рис. 3 представлен фрагмент скрипта с sql-командами.

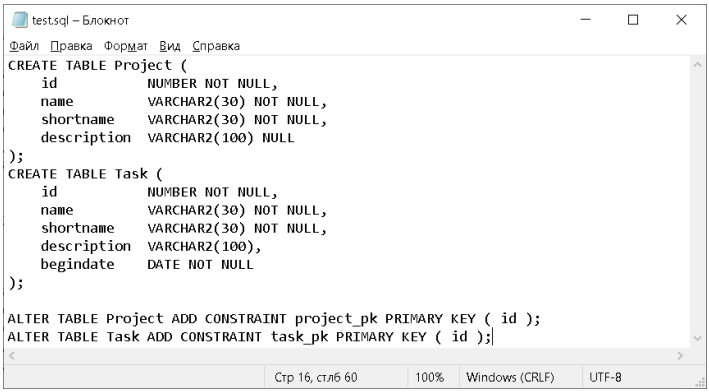

Рис. 3. Скрипт

После загрузки скрипта в приложении автоматически появятся указанные в файле таблицы и атрибуты (рис. 4).

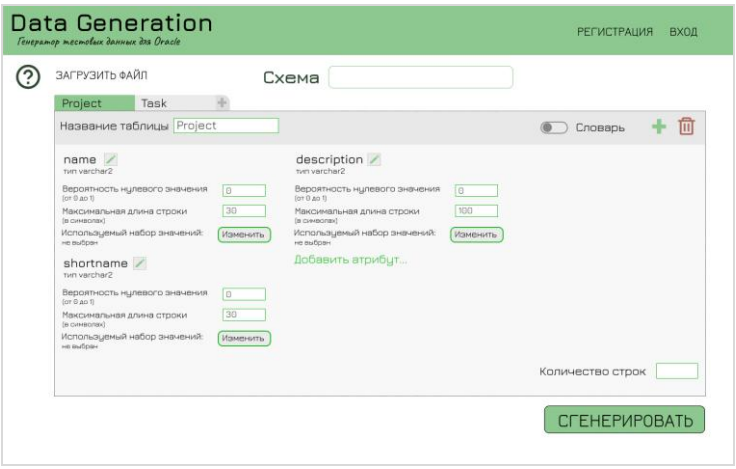

Рис. 4. Загруженные данные

Поле первичного ключа ID автоматически пропускается, так как в целях упрощения алгоритм будет автоматически создавать последовательности для каждой таблицы.

Для того чтобы добавить атрибут нужно либо нажать на кнопку «Добавить атрибут», либо на «+» в правом верхнем углу. Появится следующее модальное окно (рис. 5).

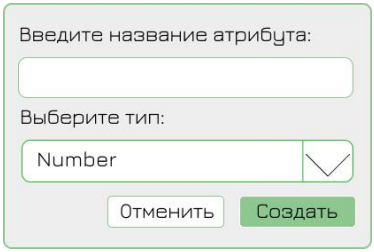

Рис. 5. Модальное окно добавления атрибута

После нажатия на кнопку «Создать» в выбранной вкладке появится созданный атрибут с указанным названием и типом.

После того, как будут указаны все необходимые параметры, требуется нажать на кнопку «Сгенерировать» и автоматически начнется скачивание файла с расширением sql, который будет содержать скрипт с заполнением таблиц.

4. Модель данных

Для реализации была выбрана нереляционная база данных. Выбор обусловлен необходимой потребностью в гибкости (также структура пользовательской базы данных заведомо неизвестна), масштабируемостью и способом хранения данных.

Кроме того, требуется обеспечить сильную вложенность, поэтому была выбрана документоориентированная база данных.

На рис. 6 представлена схема модели данных.

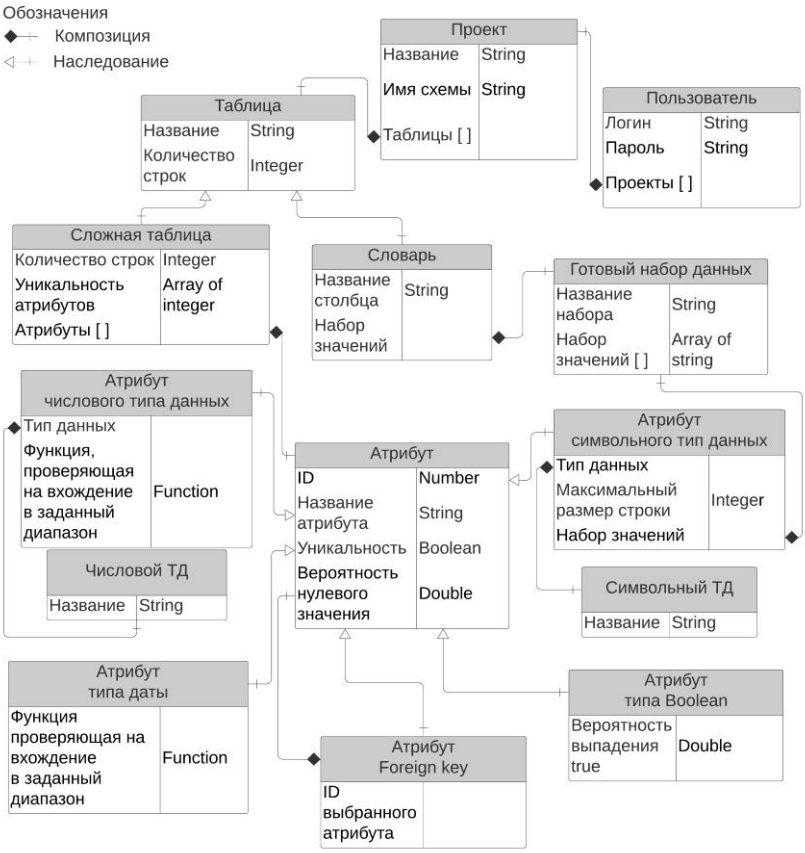

Рис. 6. Схема модели данных

В демонстрационных целях на схеме указаны типы данных.

У каждого пользователя существует набор проектов, которые он может заново использовать, редактировать, удалять, и добавлять новые. Необходимость сохранения обуславливается удобством пользователя.

Каждый проект содержит название, имя схемы и набор таблиц. Таблицы делятся на два типа: сложные (составные, состоят из нескольких атрибутов) и справочники (на схеме обе сущности наследуются от таблицы). Для заполнения их данными используются разные алгоритмы.

Сущность «Атрибут» содержит поле с информацией о первичном ключе ID генерируемой сущности (которое можно использовать в качестве внешнего ключа у других атрибутов), название атрибута, уникальность (ограничение unique) и вероятность нулевого значения (число от 0 до 1). Если стоит ограничение NOT NULL вероятность 0, если нет, то по умолчанию вероятность 0.1.

От сущности «Атрибут» наследуются несколько сущностей, которые отличаются типом данных. Это так же обусловлено потребностью в разных алгоритмах для генерации различных типов данных.

Кроме того, существуют готовые наборы данных. Они нужны для того, чтобы пользователю не приходилось каждый раз писать часто используемые наборы данных, такие как имена, фамилии, города.

5. Архитектура приложения

На рис. 7 представлена общая структура приложения.

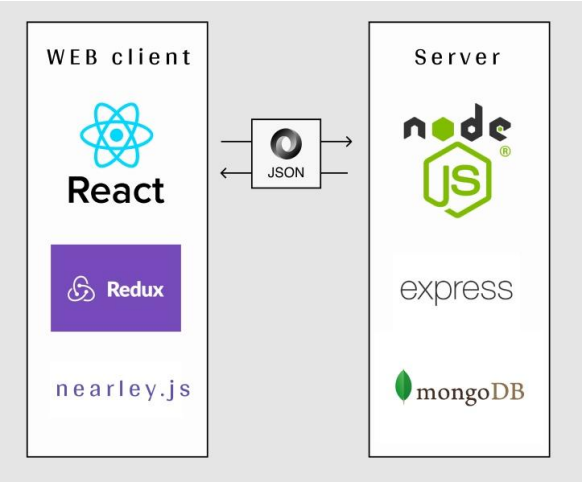

Рис. 7. Структура приложения

Приложение состоит из клиентской и серверной частей.

Клиент – веб приложение, написанное с применением JavaScript библиотеки React, с помощью которой реализуется пользовательский интерфейс.

На клиентской части происходит парсинг пользовательских данных (скриптов CREATE), реализованный с помощью библиотеки nearley.js. Парсер превращает строки символов в осмысленные структуры данных (в данном случае в JSON). В результате формируются с помощью интерфейса данные о структуре пользовательских баз данных, которые сохраняются в Redux хранилище в виде объектов JavaScript, для дальнейшего отображения и модификации.

Непосредственная генерация данных на клиентской части приложения не происходит, только задается конфигурация генерации, использующаяся сервером. Данные между клиентом и сервером передаются в понятном клиентскому приложению формате – JSON.

Сервер – приложение принимающее и обрабатывающее пользовательские запросы на генерацию данных в виде JSON, реализованное с помощью Node.js express, что позволяет использовать JavaScript на стороне сервера и не производить дополнительные преобразования полученных от пользователя данных. Сервер соединяется с базой данных mongoDB, так же использующей JSON, таким образом приложение получается полностью изоморфным. Именно на сервере осуществляется генерация и возврат данных клиенту.

На рис. 8 показана более подробная схема потока данных приложения.

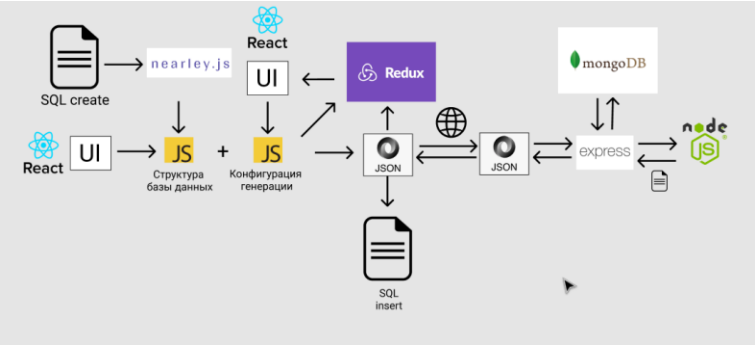

Рис. 8. Схема потока данных приложения

6. Алгоритмы генерации

Генерация данных происходит следующим образом.

Создается файл с расширением SQL, в который сначала записываются строки, создающие последовательности.

Например, для таблицы Project будет сгенерирован следующий текст:

‘CREATE SEQUENCE project_seq MINVALUE 1 MAXVALUE 99999 START WITH 1 INCREMENT BY 1;’

Название последовательности – это название таблицы + “_seq”.

Затем записываются строки с помощью шаблона: ‘INSERT INTO name (column1, column2, …, columnN) VALUES (value1, value2, … valueN);’, в который подставляются указанные пользователем значения (названия таблицы и атрибутов). Первый атрибут всегда ID, для его генерации используются последовательности, созданные выше.

Для генерации справочника, про который известно, что гарантированно, там будет только два атрибута (ID и значение), используется набор данных, выбранный или созданный пользователем. Берутся первые N строк (N указывается пользователем) из набора и записываются в вышеуказанный шаблон.

При генерации составной таблицы используются разные алгоритмы в зависимости от типа атрибута.

У всех алгоритмов есть общая черта: вероятность выпадения NULL. Вероятность указывается числом от 0 до 1. Если стоит ограничение NOT NULL, то вероятность автоматически 0. Также у различных алгоритмов есть уникальность (ограничение unique).

1. Логический тип данных

Самый простой в реализации алгоритм. Указывается только вероятность выпадения истинного значения. Затем возвращается либо ‘Y’, либо ‘N’, так как в Oracle чаще всего принято именно так реализовывать тип Boolean.

Ограничение unique не предусмотрено для логического типа.

2. Символьный тип данных

Создается копия выбранного пользователем набора значений (например, имена, фамилии, названия магазинов) и из этого набора данных удаляются те значения, длина которых превышают указанную пользователем. Затем возвращается случайное значение набора.

Если стоит ограничение unique, то на каждой итерации из набора данных удаляется выбранное значение.

3. Числовой тип данных

Пользователь выбирает тип: целое или с плавающей точкой и указать диапазон значений числа.

В дальнейшем планируется расширить набор ограничений числовых типов.

4. Тип дата и время

Пользователь может указать ограничение для даты и времени.

5. Внешние ключи

Внешний ключ это ID другой таблицы, для упрощения внешний ключ будет случайным числом от 1 до N (количества элементов в другой таблице).

В дальнейшем планируется расширить набор ограничений в зависимых таблицах, например, генерировать дату не раннее, чем дату в таблице, которая связана с первой через внешний ключ.

Заключение

По итогам работы был проведен анализ существующих приложений, на основе полученных данных была определена необходимая функциональность приложения, а также разработаны основные требования, по которым была построена модель данных приложения. Был спроектирован интерфейс сервиса для генерации тестовых данных. Также были разработаны общая архитектура приложения и алгоритмы генерации данных.

Список литературы

1. Алан Купер Интерфейс. Основы проектирования взаимодействии. 4-е издание/ Алан Купер: – Санкт-Петербург: Издательский дом «Питер», 2016. – 719 с.

2. Дейт К. Дж. Введение в системы баз данных / К.Дж. Дейт. – Москва : Диалектика, 2019 – 1328 с.

3. Мюллер, Р. Дж. Базы данных и UML. Проектирование / Р. Дж. Мюллер. − М.: ЛОРИ, 2017. − 420 с.

4. Херрон Д. Node.js Разработка серверных веб-приложений на JavaScript / Д. Хэррон ; пер. с англ. А. А. Слинкина. – Москва : ДМК-Пресс, 2016. – 144 с.