
Методика разработки инструмента генерации тестовых данных «QA Data Source»

А.А. Чумакова

*Уральский федеральный университет имени первого Президента России Б.Н. Ельцина,
Екатеринбург*

Аннотация: В статье рассматривается авторская методика проектирования и разработки инструмента генерации тестовых данных под названием «QA Data Source», которые в последующем могут быть использованы при проведении тестирования программного обеспечения. В работе описываются основные требования, функциональность приложения, модель данных, а также примеры использования. При описании приложения использовались методы системного анализа и моделирования информационных процессов. В результате применения предложенной модели реализации информационных процессов можно многократно сократить время и ресурсы на генерацию тестовых данных и последующего тестирования продукта.

Ключевые слова: обеспечение качества, тестирование ПО, тестовые данные, информационные технологии, генерация данных, базы данных, разработка приложений.

Введение

В условиях рыночной экономики для большинства компаний, занимающихся разработкой программного обеспечения (ПО), ключевое значение имеют повышение качества выпускаемого продукта и минимизация затрат на его разработку [1]. Генерация данных — это процесс создания искусственных данных, которые имитируют реальные данные и используются для различных целей, например, для тестирования программного обеспечения, обучения моделей машинного обучения или заполнения тестовых баз данных [2, 3]. Генерация тестовых данных – один из наиболее трудоемких этапов в процессе тестирования поэтому вопрос сокращения затрат на его проведение крайне актуален [4, 5].

По соображениям информационной безопасности в тестовых средах не используются реальные данные, но при этом искусственно созданные данные должны удовлетворять требованиям структуры, ограничениям и заданным бизнес-правилам [6, 7]. Таким образом, возникает необходимость использования инструмента, который бы автоматизировал логически

сложные и громоздкие процессы работы с данными, позволял искать, изменять и создавать их.

Исходя из вышесказанного, с целью оптимизации процесса тестирования был разработан генератор тестовых данных «QA Data Source», который позволяет генерировать случайные данные по заданным параметрам, что значительно упрощает и ускоряет процесс тестирования и разработки программного обеспечения.

Описание работы приложения генерации тестовых данных «QA Data Source»

«QA Data Source» доступен как при ручном тестировании через интерфейс с широким функционалом, так и для авто-тестов посредством клиентских модулей с возможностью использования технологии DataPool, то есть создания хранилища, в которое будут помещены заранее найденные данные, или сокрытие найденных данных по технологии BlackList. Рассмотрим эти технологии и концептуальные решения подробнее.

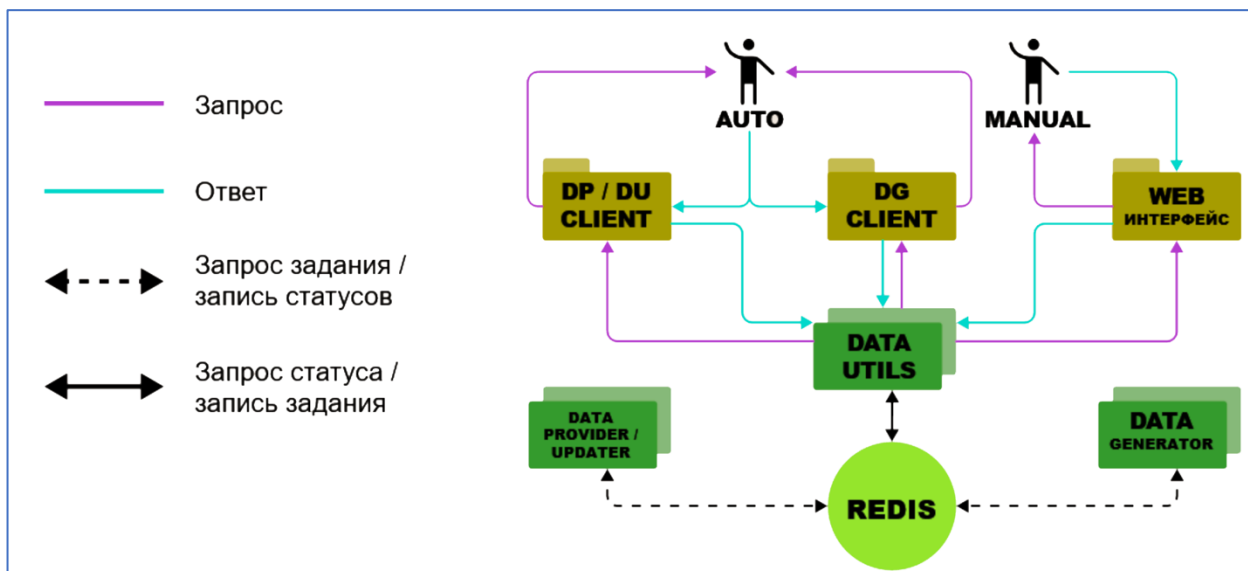
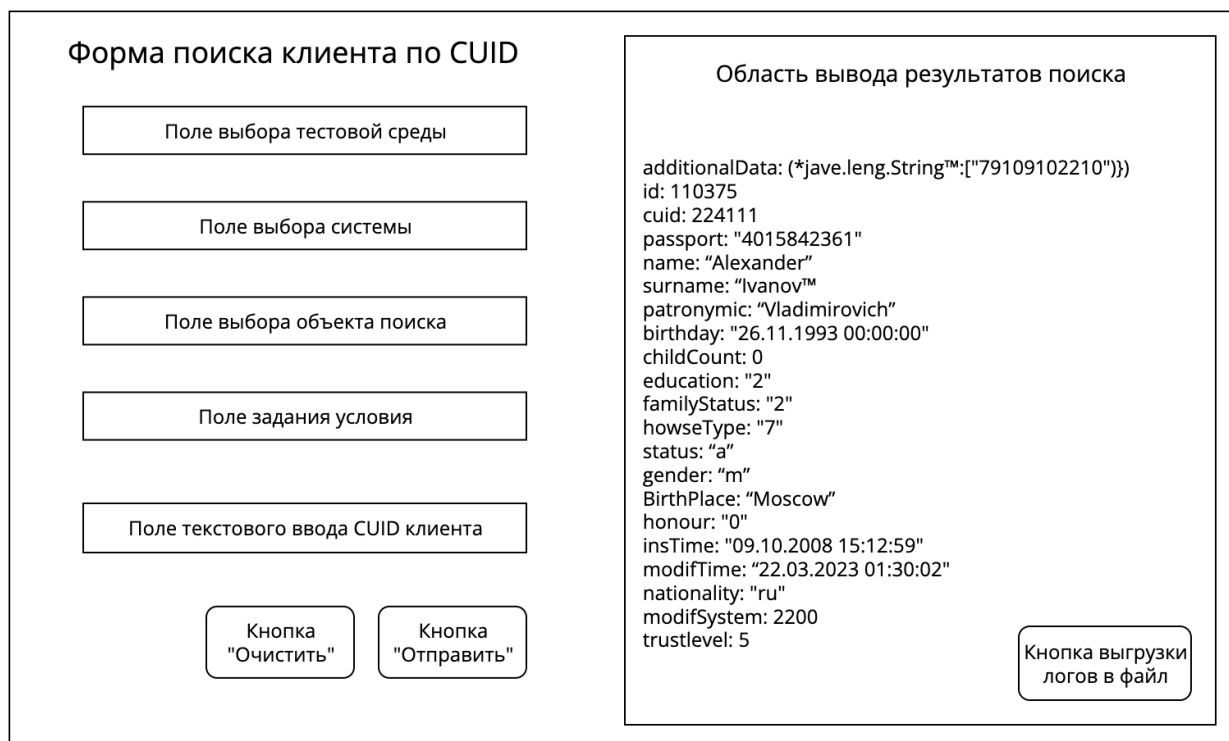


Рис. 1. – Схема взаимодействия и архитектура «QA Data Source»

Функционал реализован на базе микросервисов. При поиске и модификации данных происходит обращение SQL-скриптами или SOAP-запросами в базу данных (БД). «QA Data Source» интегрирован со всеми core-системами, которые есть в компании, и со множеством не core-систем и микросервисов [8].

Рассмотрим основную функциональность приложения на примере спроектированного интерфейса. Для поиска данных в интерфейсе необходимо выбрать тестовую среду или контур, после чего выбирается система, в которой будет осуществляться поиск данных. Что бы было удобнее ориентироваться, они сгруппированы по определенным сущностям, например, объединены все сервисы, относящиеся к банковским картам, или все сервисы, относящиеся к клиентам. Также есть возможность найти необходимый сервис по ключевому слову. Для запуска сервиса поиска необходимо указать входные параметры и нажать «Отправить».



Форма поиска клиента по CUID

Поле выбора тестовой среды

Поле выбора системы

Поле выбора объекта поиска

Поле задания условия

Поле текстового ввода CUID клиента

Кнопка "Очистить" Кнопка "Отправить"

Область вывода результатов поиска

```
additionalData: (*jave.leng.String™:["79109102210"])
id: 110375
cuid: 224111
passport: "4015842361"
name: "Alexander"
surname: "Ivanov™"
patronymic: "Vladimirovich"
birthday: "26.11.1993 00:00:00"
childCount: 0
education: "2"
familyStatus: "2"
howseType: "7"
status: "a"
gender: "m"
BirthPlace: "Moscow"
honour: "0"
insTime: "09.10.2008 15:12:59"
modifTime: "22.03.2023 01:30:02"
nationality: "ru"
modifSystem: 2200
trustlevel: 5
```

Кнопка выгрузки логов в файл

Рис. 2. – Схема формы поиска данных

В правой части мы видим выходные анонимизированные данные, а также блок с результатом поиска. Он полезен в том случае, если нужно сделать дополнительные вызовы данного сервиса или каких-либо других сервисов и иметь возможность вернуться к предыдущему. Есть возможность просмотреть или скачать журнал логирования по кнопке выгрузки логов в файл, находящейся в нижнем правом углу формы. В логах содержится информация о статусе, с которым закончил выполнение сервис, информация об используемой БД, ответ сервиса и запрашиваемые данные, в том числе информация о том, кто именно запустил сервис. Но зачастую требования к сервисам больше, чем поиск клиента, рассмотренного выше, и необходимо большее количество входных параметров. Причем какие-то параметры содержатся в БД одной системы, какие-то параметры содержатся в БД другой системы, и поиск таких данных может занимать продолжительное время, вплоть до 50-60 минут. Для решения этой проблемы был реализован адаптивный функционал DataPool, который позволяет заранее найти данные, положить их в хранилище и при очередном обращении сервиса забирать данные уже из хранилища не дольше, чем за минуту.

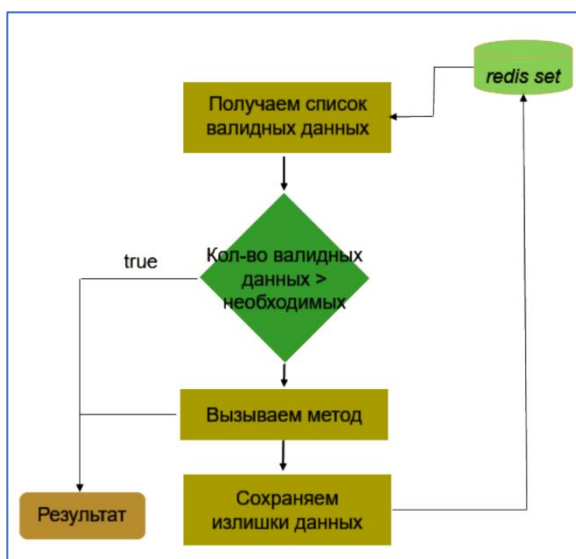


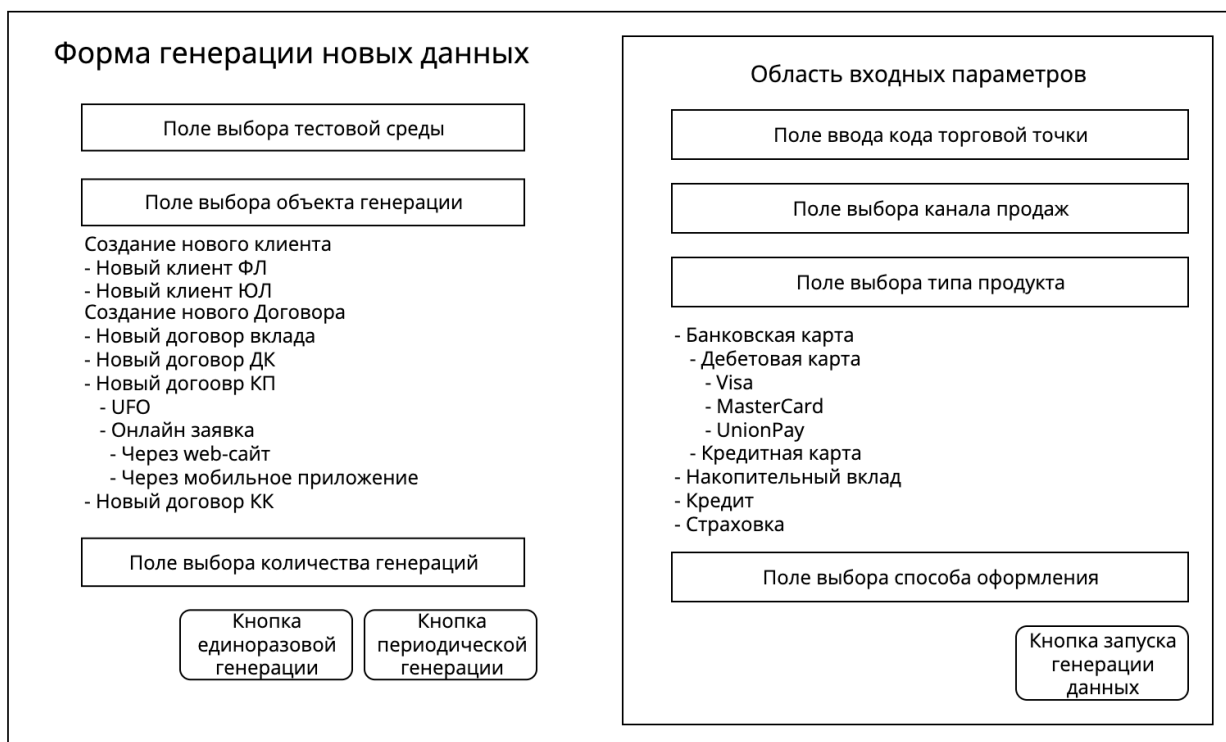
Рис. 3. – Схема работы технологии DataPool

Объем хранилища определяется в зависимости от частоты использования сервиса, с периодическими проверками не опустел ли пул и не перестали ли находящиеся там данные быть валидными.

Далее рассмотрим использование инструмента генерации данных при проведении end-to-end авто-тестов, требующих интеграции между несколькими системами [9]. Они используются в случае, если необходимо выполнить ряд действий в одной системе, получить данные, и затем переиспользовать эти данные в другой системе. Для хранения этих временных данных также используется функционал «QA Data Source», а именно временное хранилище данных, которое позволяет по определенному ключу не только положить или считать данные, но и проверить наличие этих данных. Каждый сервис имеет дополнительный функционал. Например, в вопросе, в какой именно таблице выбранной системы происходит поиск, есть возможность просмотреть используемый скрипт или же зарезервировать найденные данные пользователям и исключить из выборки для других. В этом случае можно воспользоваться функционалом Blacklist, который поможет спрятать указанные данные на указанный срок. Также есть возможность сгенерировать шаг, который в дальнейшем можно использовать как аннотацию в Gherkin-сценарий. Данный функционал по своим функциям схож с функционалом поиска, который мы рассмотрели выше, и отличается лишь по своему назначению: при «Поиске» вызывается метод SELECTED в БД, а при «Модификации» - UPDATE, при этом происходит запуск процедур, джоб и ETL-потоков, которые передают данные из системы в систему [10].

Далее рассмотрим непосредственно генерацию новых данных. Если поиск и модификация – это обращение к БД, то при генерации имитируется работа фронтальных UI систем. Логи, записанные при работе системы, используются как инструкция для создания новой генерации, и затем последовательно вызываются методы. Данная реализация наиболее

рациональна в своем использовании при поставке изменений: оформив процесс в системе, можно увидеть добавленные или измененные методы и скорректировать генерацию, а использование процедур на стороне БД затрагивает работу смежных команд, что делает ожидание доработок непрогнозируемым.



Форма генерации новых данных

Поле выбора тестовой среды

Поле выбора объекта генерации

Создание нового клиента

- Новый клиент ФЛ
- Новый клиент ЮЛ

Создание нового Договора

- Новый договор вклада
- Новый договор ДК
- Новый договор КП
- UFO
- Онлайн заявка
- Через web-сайт
- Через мобильное приложение
- Новый договор КК

Поле выбора количества генераций

Кнопка единовременной генерации

Кнопка периодической генерации

Область входных параметров

Поле ввода кода торговой точки

Поле выбора канала продаж

Поле выбора типа продукта

- Банковская карта
- Дебетовая карта
 - Visa
 - MasterCard
 - UnionPay
- Кредитная карта
- Накопительный вклад
- Кредит
- Страховка

Поле выбора способа оформления

Кнопка запуска генерации данных

Рис. 4. – Схема формы генерации новых данных

Для создания новой генерации необходимо выбрать тестовую среду и объект генерации. В правой части появляются входные параметры, они созависимы, то есть в зависимости от выбора одних параметров меняется комбинация других, например, при выборе не конечного статуса договора меняется количество возможных дальнейших манипуляций. Генерацию можно сделать сразу пачкой, указав необходимое количество, или сделать ее не в моменте, а за указанный период времени, это бывает полезно для проверки новых функций продукта, для которых требуется большое количество данных, на создание которых уходит много времени. В истории запусков помимо стандартной фильтрации отображаются задания, созданные

конкретным пользователем, или задания, созданные только от авто-тестов, а также есть возможность просмотра детальной информации.

Ниже представлена схема модели данных для генерации новых клиентов.

Каждый проект содержит название, имя схемы, набор таблиц и атрибутов каждой таблицы. У каждой созданной сущности имеется набор параметров, которые можно многократно переиспользовать, редактировать, удалять, и добавлять новые. Для удобства использования данные о каждой генерации так же сохраняются в отдельную таблицу в БД. Таблицы делятся на три типа: заполняемые SQL-скриптами, заполняемые в коннекторе (данные формируются в стриминге) и таблицы для вывода результатов в интерфейсе приложения. В основе заполнения таблиц лежат различные алгоритмы, которые обеспечивают заполнение данных в соответствии с определенными правилами и требованиями. Одним из ключевых связующих элементов таблиц является сущность "OPERATION_TYPE_ID", которая содержит идентификационную информацию об операции генерации данных. От нее наследуются различные сущности, которые отвечают за конкретные типы данных. Это обусловлено тем, что для генерации различных типов данных (например, текстовые строки, числа, даты) требуются разные алгоритмы. Например, для генерации текстовых данных используются алгоритмы, способные генерировать случайные слова, фразы, имена, адреса и т.д. Для генерации чисел применяются алгоритмы, которые могут генерировать случайные числа в заданном диапазоне, с определенной точностью и с учетом распределения (например, нормальное распределение или экспоненциальное). Генерация дат требует алгоритмов, способных генерировать случайные даты в заданном интервале времени.

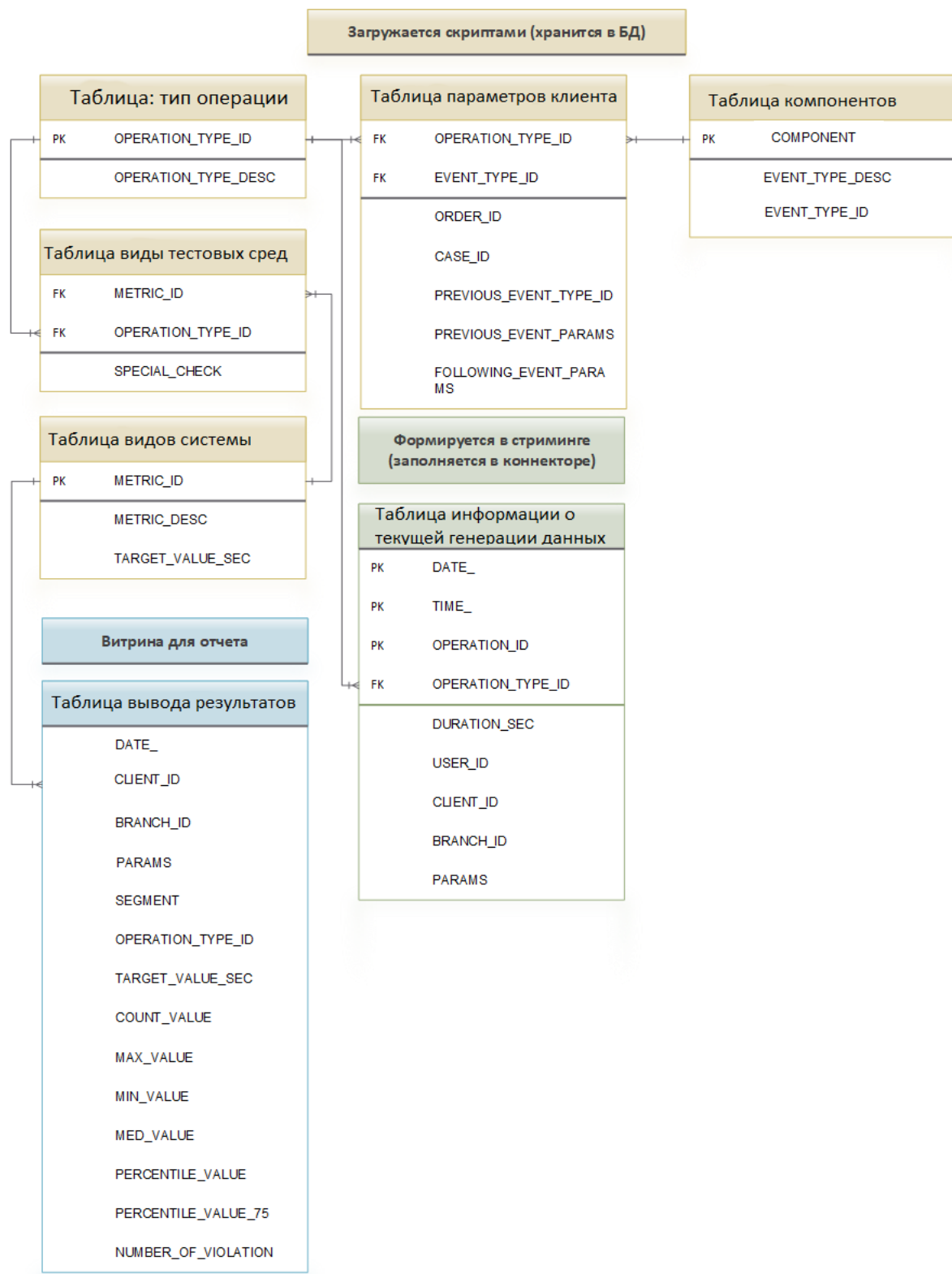


Рис. 5. – Модель данных на примере создания новых клиентов

Так же в «QA Data Source» используются готовые наборы данных. Они представляют собой предустановленные наборы данных, включающие часто используемую информацию, такую как:

Имена - список распространенных имен (как мужских, так и женских), которые могут использоваться для заполнения столбцов с именами.

Фамилии - список распространенных фамилий, который может быть использован для заполнения столбцов с фамилиями.

Города - список городов, который может быть использован для заполнения столбцов с местом жительства.

Другие данные - готовые наборы данных могут включать также другие типы данных, например, адреса, телефонные номера, электронные адреса и так далее. Использование готовых наборов данных значительно упрощает процесс заполнения таблиц, позволяет сократить время тестирования и повысить эффективность работы [11].

Рассмотрим подробнее алгоритмы генерации данных.

Для генерации данных используется скрипт на языке SQL, который позволяет создавать и заполнять таблицы. Первым шагом является создание последовательностей, которые нужны для генерации уникальных идентификаторов (ID) для записей в таблицах. Последовательность — это специальный объект базы данных, который автоматически генерирует возрастающие числа. Например, для «таблицы параметров клиента» будет создана последовательность с помощью следующего SQL-запроса [12]:

```
CREATE SEQUENCE clients_seq AS BIGINT  
START WITH 1  
INCREMENT BY 1  
MINVALUE 1  
MAXVALUE 99999  
NO CYCLE  
CACHE 10;
```

После создания последовательностей можно приступать к заполнению таблиц данными. Для этого используется команда `INSERT INTO`, которая позволяет вставить новые записи в таблицу. Вставка записей происходит по следующему шаблону:

```
INSERT INTO имя_таблицы (поле1, поле2, ...) VALUES  
(значение_поля_1, значение_поля_2, ...);
```

Первый столбец в таблице обычно является столбцом ID, для его генерации используется созданная ранее последовательность. Значения для остальных столбцов могут быть заданы пользователем или сгенерированы автоматически с помощью различных алгоритмов, которые будут рассмотрены ниже.

У всех алгоритмов есть возможность генерировать пустые значения (NULL). Вероятность выпадения NULL указывается числом от 0 до 1. Если для столбца задано ограничение «NOT NULL», то вероятность выпадения NULL автоматически устанавливается равной 0. Некоторые алгоритмы обеспечивают уникальность генерируемых значений. Например, для столбца «email» можно указать ограничение «unique», чтобы гарантировать, что каждый сгенерированный адрес электронной почты будет уникальным.

Рассмотрим типы алгоритмов, используемых в «QA Data Source»:

1. Алгоритм генерации логических данных (true/false) – простой алгоритм, в котором указывается только вероятность выпадения истинного значения (например, 0.7 для вероятности 70%). Затем генератор возвращает либо `Y` (true), либо `N` (false), так как БД часто используются именно эти символы для представления логических значений. Ограничение «unique» для логического типа не предусмотрено, так как нет необходимости в уникальности булевых значений.

2. Генерация данных символьного типа (например, имена, фамилии, названия магазинов) в которой используется набор значений, предоставленный пользователем. Алгоритм генерации символьных данных может быть реализован различными способами, например, с использованием случайного выбора из предоставленного набора: алгоритм случайным образом выбирает значения из предоставленного списка, например, можно использовать случайный выбор имени и случайную генерацию фамилии.

3. Для генерации числовых данных (например, возраста, зарплаты, цен) используются различные алгоритмы, которые могут учитывать: диапазон значений (генерация случайных чисел в заданном диапазоне, например, от 18 до 65 для возраста), распределение вероятности (учет различных распределений вероятности, например, нормальное распределение для зарплаты), взаимосвязь с другими данными (учет взаимосвязи между числовыми данными в таблице, например, между возрастом и зарплатой).

4. Для генерации дат и времени используются алгоритмы, которые могут генерировать случайные даты и время в заданном диапазоне, учитывать различные форматы дат и времени, генерировать даты с учетом различных событий, например, праздников.

Заключение

В результате исследования, были сделаны выводы о том, что генерация данных — это важный процесс, который позволяет создавать искусственные данные для тестирования ПО. Важно понимать, что генерируемые данные не всегда должны быть идеально точными, главное, чтобы они были релевантными и удовлетворяли поставленным задачам. В данной статье были описаны основные требования, модель данных, спроектирован интерфейс, разработана общая архитектура и обозначены используемые алгоритмы генерации данных инструмента «QA Data Source».

«QA Data Source» может выступать как единая точка входа для тестировщиков, аналитиков, разработчиков и прочих стейкхолдеров. Он выступает в роли централизованного управления данными и следит за актуальностью сервисов, обеспечивая прозрачную работу с данными, а также позволяет сократить трудозатраты тестирования за счет функционала DataPool и Blacklist, а также автоматизировать интеграционные проверки за счет временного хранилища данных.

Литература

1. Kathleen F., Terence P. LL(*): the foundation of the ANTLR parser generator // Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI). 2011. pp. 325-336.
2. Гинис Л.А. Методологические основы нечеткого когнитивного моделирования иерархических проблемно-ориентированных систем // Инженерный вестник Дона, 2014, №2. URL: ivdon.ru/ru/magazine/archive/n2y2014/2326/
3. Дейт К. Дж. Введение в системы баз данных. Москва: Диалектика, 2019. 347 с.
4. Херрон Д. Node.js Разработка серверных веб-приложений на JavaScript. Москва: ДМКПресс, 2016. 91 с.
5. Избачков Ю.С., Петров В.Н., Васильев А.А., Телина И.С. Информационные системы. Спб.: Издательский дом «Питер», 2021. 244 с.
6. Колесникова Д.С., Верещагина Е.А., Гуляев В.Е. Построение онтологической модели для предметной области «Информационная безопасность» // Инженерный вестник Дона, 2023, №7. URL: [ivdon.ru/ru/magazine/archive/n7y2023/8532.](http://ivdon.ru/ru/magazine/archive/n7y2023/8532/)
7. Duda R.O., Hart P.E., Stork D.G. Pattern classification and scene analysis. New York: Wiley, 1973. 482 p.

8. Мюллер Р. Дж. Базы данных и UML. Проектирование. М.: ЛОРИ, 2017. 388 с.
9. Disting E., Rashka J., Paul J. Automated Software Testing // ADDISON-WESLEY. 2017. pp. 46-47.
10. Murad D.F., Heryadi Y., Wijanarko B.D., Isa S.M., Budiharto W. Recommendation System for Smart LMS Using Machine Learning: A Literature Review // International Conference on Computing, Engineering, and Design (ICCED). 2018. pp. 36-38.
11. Cormen T.H., Leiserson Ch.E., Rivest R.L., Stein C. Introduction to Algorithms. The MIT Press, 2009. 1292 p.
12. Купер А. Интерфейс. Основы проектирования взаимодействия. Санкт-Петербург: Издательский дом «Питер», 2016. 615 с.

References

1. Kathleen F., Terence P. Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI). 2011. pp. 325-336.
 2. Ginis L.A. Inzhenernyj vestnik Dona, 2014, №2. URL: ivdon.ru/ru/magazine/archive/n2y2014/2326/
 3. Deyt K. Dzh. Vvedenie v sistemy baz dannykh [Introduction to database systems]. Moskva: Dialektika, 2019. 347 p.
 4. Kherron D. Node.js Razrabotka servernykh veb-prilozheniy na JavaScript [Node.js Development of server-side web applications in JavaScript]. Moskva: DMKPress, 2016. 91 p.
 5. Izbachkov Yu.S., Petrov V.N., Vasil'ev A.A., Telina I.S. Informatsionnye sistemy [Information systems]. Spb.: Izdatel'skiy dom «Piter», 2021. 244 p.
 6. Kolesnikova D.S., Vereshchagina E.A., Gulyaev V.E. Inzhenernyj vestnik Dona, 2023, №7. URL: ivdon.ru/ru/magazine/archive/n7y2023/8532.
-



7. Duda R.O., Hart P.E., Stork D.G. Pattern classification and scene analysis. New York: Wiley, 1973. 482 p.
8. Myuller R. Dzh. Bazy dannykh i UML. Proektirovanie [Databases and UML. Designing]. M.: LORI, 2017. 388 p.
9. Disting E., Rashka J., Paul J. ADDISON-WESLEY. 2017. pp. 46-47.
10. Murad D.F., Heryadi Y., Wijanarko B.D., Isa S.M., Budiharto W. International Conference on Computing, Engineering, and Design (ICCED). 2018. pp. 36-38.
11. Cormen T.H., Leiserson Ch.E., Rivest R.L., Stein C. Introduction to Algorithms. The MIT Press, 2009. 1292 p.
12. Kuper A. Interfeys. Osnovy proektirovaniya vzaimodeystvii [Basics of interaction design]. Sankt-Peterburg: Izdatel'skiy dom «Piter», 2016. 615 p.

Дата поступления: 3.07.2024

Дата публикации: 16.08.2024