

# Алгоритмы генерации тестовых данных

Тишенинов М.А., СПбГУ, Санкт-Петербург [st096244@student.spbu.ru](mailto:st096244@student.spbu.ru),  
Благов М.В., СПбГУ, Санкт-Петербург [st024103@student.spbu.ru](mailto:st024103@student.spbu.ru)

## Аннотация

В данной статье приводится адаптация алгоритма DSOMA для генерации данных для тестирования SQL-запросов, поддерживающий все виды соединений, а также агрегационные запросы и оконные функции. В работе также приводится сравнительный анализ с другими алгоритмами, решающими такую задачу.

## Введение

Формирование тестовых данных — одна из ключевых задач, возникающих в процессе тестирования программного обеспечения, и она сохраняет свою актуальность на протяжении многих лет [1, 2]. Это особенно важно в контексте тестирования взаимодействий с базами данных [3].

При тестировании запросов к базам данных разработчики сталкиваются с рядом вызовов, поскольку любые значимые изменения в программном обеспечении, взаимодействующем с базами данных, обычно приводят к усложнению SQL-запросов. С ростом требований к функциональности и производительности систем, запросы становятся все более многоуровневыми, включают сложные фильтры, объединения таблиц, агрегации и группировки. Рост сложности запросов также влечёт за собой увеличение временных затрат на получение тестовых данных для их тестирования [2].

Одним из популярных подходов к получению тестовых данных является их генерация [4, 5], она позволяет быстро получить необходимый набор данных для тестирования программы и при этом не несёт рисков раскрытия конфиденциальной информации, являющихся существенным ограничением для использования реальных данных при тестировании.

## Постановка задачи

### *SQL-запрос*

В процессе исполнения SQL-запроса любая СУБД формирует план его исполнения, который можно рассматривать как программу  $P$ . Входными данными такой программы будут являться данные, удовлетворяющие ограничениям, которые заданы типами данных и связями между таблицами в базе данных, обозначим всё множество входных данных за  $D$ . У SQL-запроса имеется

набор сценариев, по которому может пойти его исполнение, обозначим этот набор за  $S$ . В результате работы программы над некоторыми исходными данными  $T \subset D$  получаем выходные данные, обозначим их  $P(T)$ .

### *Тестовые данные*

Для тестирования SQL-запроса необходим некоторый набор данных, который с одной стороны позволяет проверить большое число сценариев из  $S$ , но одновременно не является слишком громоздким. Обозначим набор тестовых данных как  $T$ , причём  $T \subseteq D$ , а также  $|T| < +\infty$ .

Для оценки качества тестовых данных будем набор функций  $M$ . Формально их можно определить как отображения  $\mu \in M : T \rightarrow \mathbb{R}^+$ . Значением функции  $\mu \in M$  на наборе тестовых данных  $T$  будет являться количество шагов в плане исполнения сценария  $s \in S$ , которое осталось пройти после получения нулевого результата. При нулевом значении функции результат работы программы на определённом сценарии будет не пуст, а значит в наборе данных существует подмножество, которое позволит протестировать исходный сценарий. Таким образом с каждым рассматриваемым сценарием  $s \in S$  связана функция  $\mu_s$ .

### *Генерация тестовых данных*

Известно множество сценариев  $S$ , множество функций  $M$ , связанных со сценариями и ограничения которым должны удовлетворять входные данные, то есть известно множество  $D$ .

Сформулируем задачу генерации тестовых данных для SQL-запроса в виде задачи оптимизации:

$$F(T) = \lambda|T| + \sum_{s \in S} \mu_s(T) \rightarrow \min_{T \in 2^D} \quad (1)$$

где  $2^D$  — множество всех возможных подмножеств множества  $D$ ,  $\lambda$  — параметр, отвечающий за значимость размера набора тестовых данных.

Задача генерации тестовых данных сведена к задаче минимизации и поэтому можем применять методы решения оптимизационных задач, в том числе и генетические алгоритмы.

### ***Покрытие тестовых сценариев***

Покрытие тестовых сценариев будем использовать в качестве метрики эффективности работы алгоритмов генерации тестовых данных. Оно будет измеряться в процентах, а вычисляться как умноженное на 100 отношение числа сценариев  $s \in S$ , для которых результат исполнения на наборе тестовых данных будет непустым к общему числу сценариев.

Получение всего множества сценариев исполнения  $S$  для SQL-запроса производится с помощью процедуры SQLFpc[6], которая подразумевает разбиение SQL-запроса на предикаты, обращение всех полученных предикатов и компоновку новых SQL-запросов из предикатов SQL-запроса и обращённых предикатов.

## **Алгоритм генерации тестовых данных**

### ***Принцип работы DSOMA***

DSOMA [7] принадлежит к классу генетических алгоритмов, их суть заключается в том, что минимум целевой функции приближается с помощью последовательных итераций над набором данных, имитирующих естественный отбор.

Алгоритм в качестве входных данных принимает набор из  $M$  векторов из  $\mathbb{N}^n$ , которое обычно называют начальной популяцией, обозначим её за  $X_0$ . Также для работы алгоритма необходима некоторая функция  $F : \mathbb{N}^{n \times M} \rightarrow \mathbb{R}^+$ , с помощью которой может быть определено отношение порядка между элементами множества входных данных.

Итерационный процесс начинается с сортировки популяции  $X_k$ , для этого необходимо вычислить значения функции  $F$  на каждом её элементе и затем отсортировать эти элементы по убыванию значения  $F$ . Первый элемент в отсортированной популяции назовём лидером.

Для всех остальных элементов популяции, кроме лидера, проведём следующую процедуру, описанную в Алг. 1.

Алгоритм 1. Создание новых элементов популяции.

```
1  FOR i = 2, 3, ..., M DO:
2    FOR j = 1, 2, ..., n DO:
3      J[j] = abs(X_k[1, j] - X_k[i, j]);
4    J_max = mode(J);
5    IF J_max >= J_min:
6      s = ceil(J_max / J_min);
```

```

7   ELSE:
8       s = 1;
9   FOR j = 2, 3, ..., M DO:
10      FOR l = 1, 2, ..., J_min DO:
11          IF X_k[i, j] < X_k[1, j]:
12              G[1, j] = X_k[i, j] + s * 1;
13          ELSE IF X[i, j] > X[1, j]:
14              G[1, j] = X_k[i, j] - s * 1;
15          ELSE:
16              G[1, j] = 0;

```

После выполнения процедуры для всех элементов популяции получим набор множеств  $G_i$ , состоящих из векторов, которые "двигаются" в направлении лидера. Объединим  $X_k$  и все  $G_i$ , после чего вычислим для значения функции  $F$  и отсортируем по возрастанию значения функции  $F$  получившееся множество, после этого удалим из него все элементы кроме первых  $M$ , тем самым получив новую популяцию  $X_{k+1}$ .

### ***Адаптация DSOMA для генерации тестовых данных***

Можно заметить, что DSOMA не подходит для решения задачи генерации тестовых данных, так как множеством в котором ведётся поиск решения в задаче является множество всех подмножеств множества  $D$  или для краткости  $2^D$  в то время как DSOMA ведёт поиск решения в пространстве  $\mathbb{N}^{n \times M}$ . Поэтому для его применения надо свести рассматриваемую задачу к задаче, которую может решить алгоритм. Для этого необходимо было сделать следующие модификации.

Во-первых, необходимо ограничить множество входных данных  $D$ , введём для этого параметр  $a \in \mathbb{N} : |D_a| = a$ . Это позволит установить биекцию между пространством индексов записей в таблицах  $\mathbb{N}^a$  и пространством  $D_a$  записей в  $l$  таблицах, необходимых для рассматриваемого запроса, так как элементами  $D_a$  в случае SQL-запроса является конечное число записей в таблицах, которые можно проиндексировать. Записи в таблицах индексируются независимо для каждой таблицы таким образом, чтобы при чтении по порядку индексов они были отсортированы в лексикографическом порядке. Набором тестовых данных в таком случае будем называть матрицу  $T$ , которая состоит из  $l$  векторов  $\mathbb{N}^n$ , где  $n \leq \frac{a}{l}$  содержащих индексы записей для каждой таблицы, используемой в SQL-запросе, таким образом  $T = (t_{ij})_{i=1, j=1}^{n, l}$ , где  $t_{ij} \in \mathbb{N}$ .

Во-вторых, необходимо определить функцию  $F$ , с помощью которой бу-

дем определять отношение порядка на множестве входных данных. Для этого будем использовать целевую функцию  $F(T) = \mu_s(T)$ , где  $s \in S$ , подобная функция уже встречалась в формулировке (1).

Процедура генерации начинается с парсинга рассматриваемого SQL-запроса, это позволяет выделить все константы, которые используются в этом запросе и атрибуты с которыми они взаимодействуют. После этого начинается генерация данных, однако используя только что полученные отношения констант в запросе и атрибутов при генерации записей в таблицах производится подстановка этих констант в сгенерированные элементы, что гарантирует наличие элементов с таким значением, а также наличие элементов с другими значениями. В каждой таблице, задействованной в SQL-запросе генерируется равное число записей.

Таким образом можем найти  $T_s^*$  с помощью модифицированного DSOMA, для каждого доступного сценария из  $S$  и после этого объединить эти наборы данных в единый набор  $T^*$ , который будет учитывать все возможные сценарии. Количество данных в каждом из  $T_s^*$  регулируется параметром  $n$  и поэтому в ходе работы алгоритма не производится оптимизация по размеру  $T^*$ .

## Эксперименты

### Параметры

Эксперименты проводились на наборе из 40 разнообразных SQL-запросах, взятых из системы управления взаимодействиями с клиентами EspoCRM [8]. Запросы содержат различные виды соединения таблиц, агрегационные запросы и оконные функции.

Основными параметрами алгоритма генерации тестовых данных, помимо SQL-запроса и схемы необходимых для него данных, являются  $M$  — размер популяции,  $a$  — размер множества входных данных,  $J$  — количество новых векторов, получаемых на каждой итерации алгоритма для каждого элемента популяции.  $K$  — количество новых векторов, добавляемое на каждой итерации алгоритма для сохранения разнообразия, а также  $n$  — количество записей из каждой таблицы  $D_a$ , индексы которых находятся в элементах  $T$ .

### Результаты

Об эффективности работы можно судить по совокупности двух показателей — времени работы алгоритма и проценту покрытия сценариев. Для на-



сравнение разработанного алгоритма с существующими аналогами, по итогам которого было выяснено, что разработанный алгоритм позволяет добиваться хорошего покрытия сценариев на 44% быстрее, чем ближайший аналог.

### Список литературы

- [1] Chan M., Shing C., Testing Database Applications with SQL Semantics // International Symposium on Cooperative Database Systems for Advanced Applications, 1999
- [2] C. Yan, S. Nath, S. Lu, Generating Test Databases for Database-Backed Applications // 2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE), Melbourne, Australia, 2023, pp. 2048-2059
- [3] Haller K., The test data challenge for database-driven applications // Proceedings of the Third International Workshop on Testing Database Systems (DBTest), Article 6, 1–6.
- [4] Baudry B. et al. Generative AI to Generate Test Data Generators // IEEE Software, vol. 41, no. 6, pp. 55-64
- [5] Patki N., Wedge R., Veeramachaneni K. The Synthetic Data Vault // 2016 IEEE International Conference on Data Science and Advanced Analytics (DSAA), 2016, pp. 399-410
- [6] Tuya J., Suárez-Cabal M., María José, de la Riva C., Full predicate coverage for testing SQL database queries // Software Testing, Verification Reliability, Volume 20, Issue 3, 2010, pp. 237-288
- [7] Davendra D., Zelinka I., Pluhacek M., Senkerik R. DSOMA—Discrete Self Organising Migrating Algorithm // Self-Organizing Migrating Algorithm. Studies in Computational Intelligence, vol 626. Springer, pp 51-63.
- [8] EspoCRM. Исходный код: [Электронный ресурс] // URL: <https://github.com/espocrm/espocrm> (Дата обращения: 06.03.2025)
- [9] Castelein J., Aniche M., Soltani M., Panichella A., van Deursen A. Search-based test data generation for SQL queries // Proceedings of the 40th International Conference on Software Engineering, Association for Computing Machinery, pp 1220–1230.