Московский авиационный институт (национальный исследовательский университет)
Институт № 8 «Компьютерные науки и прикладная математика»
Кафедра № 806 «Вычислительная математика и программирование»

Проектирование системы переноса и генерации взаимосвязанных данных из производственной среды при тестировании образовательной платформы

Выпускная квалификационная работа бакалавра

Студент группы М8О-406Б-21: Мезенин Олег Александрович
Научный руководитель: ст. преподаватель кафедры 806
Миронов Евгений Сергеевич

Москва — 2025

Актуальность темы

- Тестирование программного продукта важно.
- Тестирование нельзя проводить в производственной среде:
  - лишняя нагрузка на систему;
  - юридические риски.
- Копирование всех данных:
  - занимает много времени;
  - использует много ресурсов.
- Для многих тестовых сценариев не нужно много данных, но нужны согласованные данные.

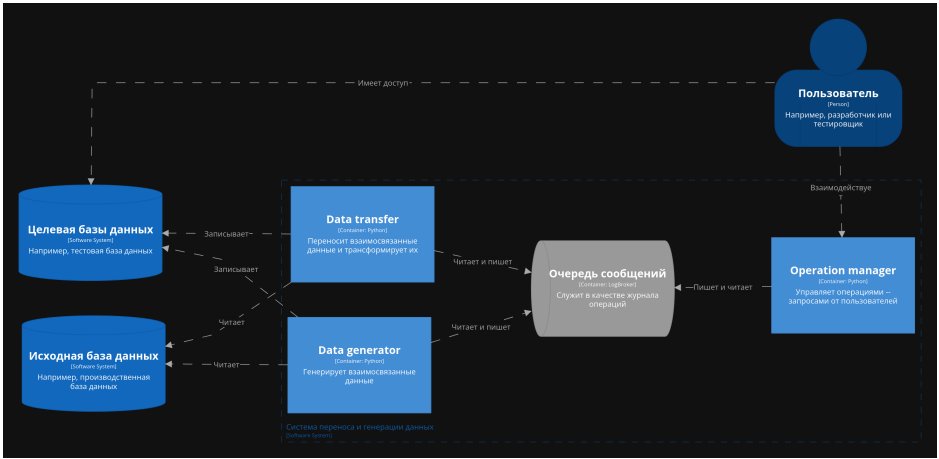
Цель и задачи работы

Цель — проектирование системы, способной переносить взаимосвязанные данные, анонимизировать их и генерировать тестовые данные.

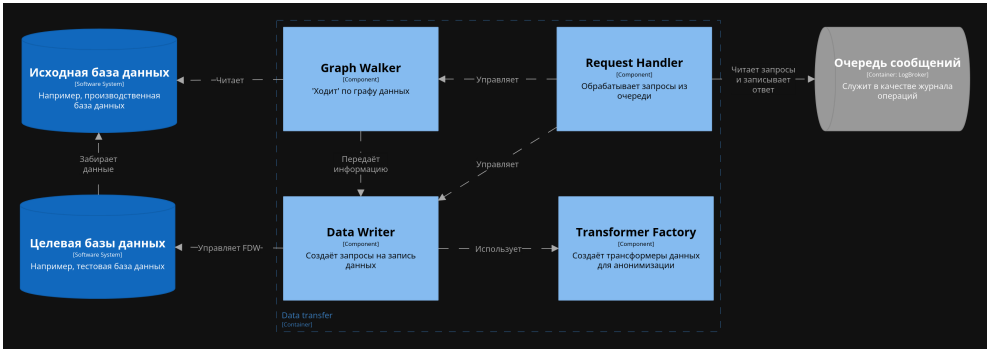
Задачи:

1. определение требований;
2. анализ аналогов;
3. проектирование архитектуры;
4. разработка алгоритмов переноса и генерации;
5. разработка языка для описания данных;
6. реализация минимально жизнеспособный продукт;
7. анализ результатов.

Система переноса и генерации данных



Компоненты Data Transfer



Предпосылки к использованию метаграфов:

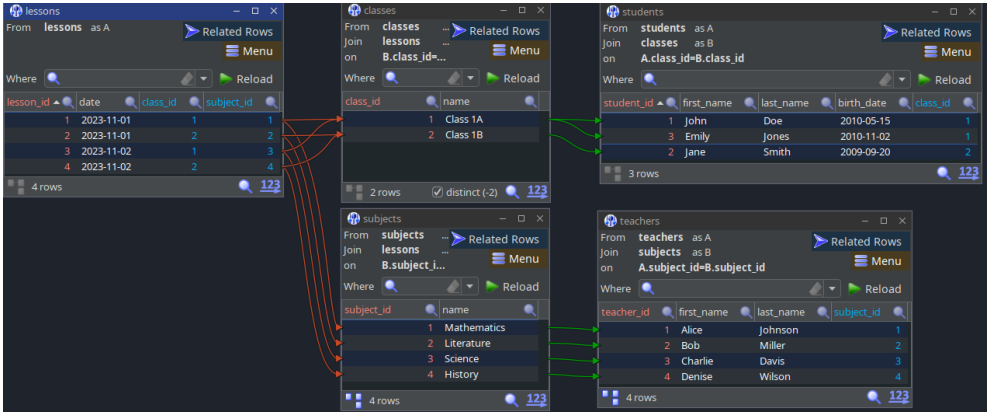
- система данных в БД и их взаимосвязей напоминает графовую структуру;
- классические графы и ER-модель не подходят ввиду ограниченных возможностей.

Пусть $MG = \langle V, MV, E, ME \rangle$ – метаграф, где V – множество вершин, MV – множество метавершин, E – множество рёбер, ME – множество метарёбер.

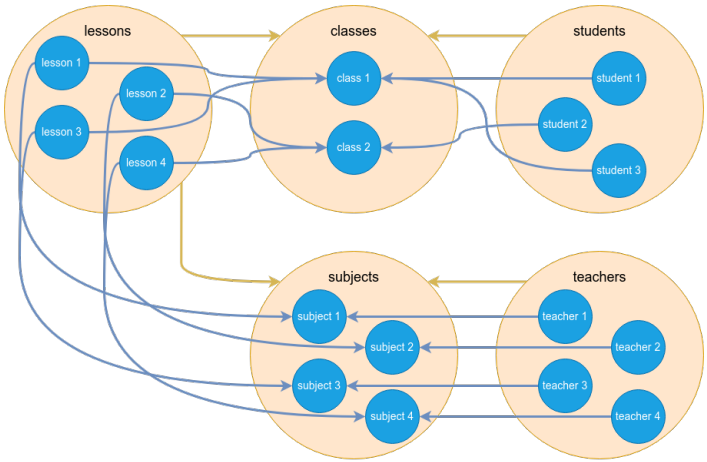
Соответствие объектов в метаграфе и БД:

- вершина $\Leftrightarrow$ запись;
- метавершина $\Leftrightarrow$ таблица;
- ребро $\Leftrightarrow$ связь между записями;
- метаребро $\Leftrightarrow$ логический внешний ключ.

Пример базы данных



Графическое отображение метаграфа



Пусть $r = \langle p, f \rangle$ – правило метаграфа, где p — предикат, а f — функция, изменяющая множества метаграфа

```
1  Base_Metagraph_Traversal(DB, SV)
2    queue ← ∅
3    visited ← ∅
4    for each v in SV
5      do Enqueue(queue, v)
6    while queue ≠ ∅
7      do cur_v ← Dequeue(queue)
8         Add(visited, cur_v)
9         for each u in Adjacent(DB, cur_v)
10          do if u ∉ visited
11             then Enqueue(queue, u)
12   return visited
```

Алгоритм обхода данных


```
1  Metagraph_Traversal_With_Rules(DB, SV, RME, RE)
2    MG ← INIT(DB)
3    MG.ME ← Apply_Rules(MG.ME, RME)
4    queue ← ∅
5    visited ← ∅
6    for each v in SV
7      do Enqueue(queue, v)
8    while queue ≠ ∅
9      do cur_v ← Dequeue(queue)
10        Add(visited, cur_v)
11        Add(MG.V, cur_v)
12        incident_edges ← Incident(DB, cur_v)
13        new_incident_edges ← Apply_Rules(incident_edges, RE)
14        Extend(MG.E, new_incedent_edges)
15        for each u in Adjacent(MG, cur_v)
16          do if u ∉ visited
17             then Enqueue(queue, u)
18   return visited
```

Примеры грамматических конструкций на ANTLR4

```
1 graph_source_stmt: GRAPH_ SOURCE_ table_name (WHERE_ expr)?;
2 no_enter_stmt: NO_ ENTER_ table_name (WHERE_ expr)?;
3 no_exit_stmt: NO_ EXIT_ table_name (WHERE_ expr)?;
4 limit_visits_stmt: LIMIT_ VISITS_ INTEGER_LITERAL FOR_
    table_name;
5 transformer_stmt: TRANSFORMER_ function_call FOR_ table_name
    DOT column_name (COMMA table_name DOT column_name)*;
```

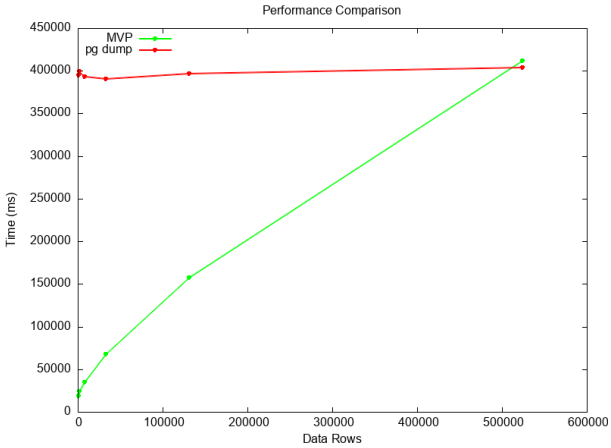
Пример описания метаданных для переноса данных

```
1 GRAPH SOURCE classes WHERE class_id=1;
2 NO ENTER teachers;
3 TRANSFORMER random_first_name FOR students.first_name;
4 TRANSFORMER random_last_name FOR students.last_name;
```

Возможности:

- интерфейс командной строки;
- перенос взаимосвязанных данных;
- поддержка правил метаграфа:
  - *NO EXIT*;
  - *NO ENTER*;
  - *LIMIT DISTANCE*.

Тестовая база данных: 4.5 ГБ, 44739072 записей



Результаты:

- спроектирована архитектура, направленная на отказоустойчивость;
- разработан язык описания метаданных;
- разработан минимально жизнеспособный продукт.

Дальнейшие перспективы:

- улучшение производительности;
- разработка анонимизации и генерации данных.

Ссылки на код грамматики и код минимально жизнеспособного продукта