

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



- Тестирование программного продукта является важной частью разработки.
- Тестирование в производственной среде сопряжено с рисками:
  - нагрузка на систему;
  - целостность данных;
  - конфиденциальность данных.
- Копирование всех данных (например, с помощью `pg_dump`):
  - занимает много времени;
  - использует много ресурсов;
  - не решается вопрос конфиденциальности данных.
- Часто для тестовых сценариев нужны не все данные, а определённое количество согласованных данных.

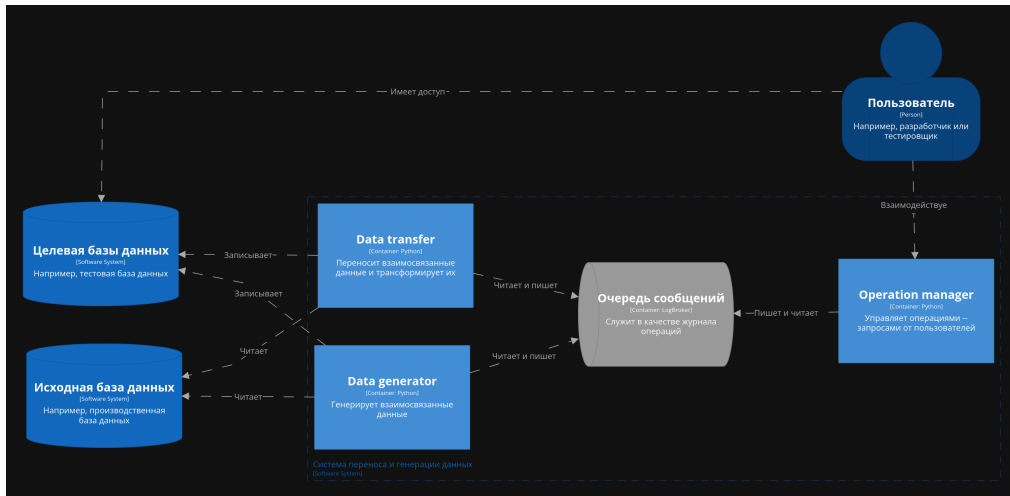


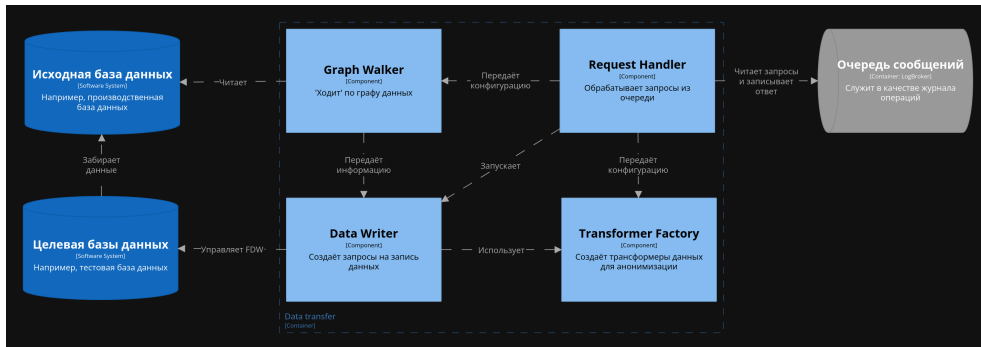
**Цель** — проектирование системы, способной переносить взаимосвязанные данные, анонимизировать их и генерировать тестовые данные.

**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 жизнеспособного продукта;
- 7 анализ результатов.







Предпосылки к использованию метаграфов:

- требовалось формализовать схему базы данных;
- классические графы и ER-модель не подходят ввиду ограниченных возможностей.

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



- вершины  $\Leftrightarrow$  записи;
- метавершины  $\Leftrightarrow$  таблицы;
- рёбра  $\Leftrightarrow$  связи между записями;
- метарёбра  $\Leftrightarrow$  логические внешние ключи.



The screenshot displays a database management interface with five tables and their relationships:

- lessons** (4 rows):

lesson_id	date	class_id	subject_id
1	2023-11-01	1	1
2	2023-11-01	2	2
3	2023-11-02	1	3
4	2023-11-02	2	4
- classes** (2 rows):

class_id	name
1	Class 1A
2	Class 1B
- students** (3 rows):

student_id	first_name	last_name	birth_date	class_id
1	John	Doe	2010-05-15	1
3	Emily	Jones	2010-11-02	1
2	Jane	Smith	2009-09-20	2
- subjects** (4 rows):

subject_id	name
1	Mathematics
2	Literature
3	Science
4	History
- teachers** (4 rows):

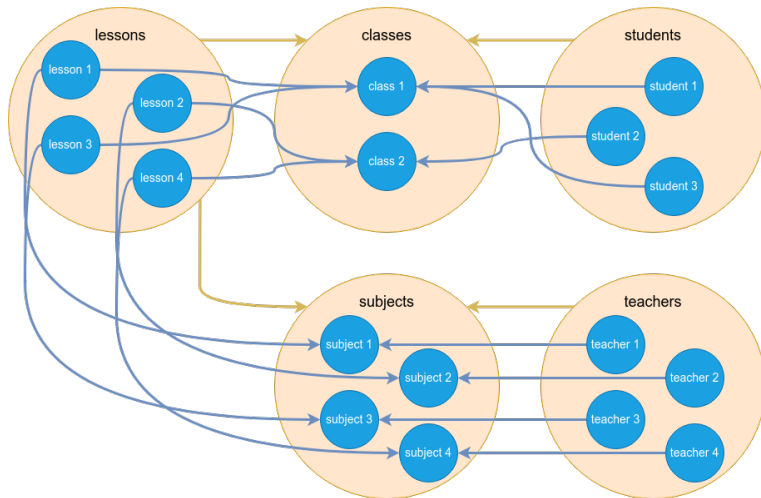
teacher_id	first_name	last_name	subject_id
1	Alice	Johnson	1
2	Bob	Miller	2
3	Charlie	Davis	3
4	Denise	Wilson	4

Relationships indicated by arrows:

- lessons** to **classes**: `class_id`
- lessons** to **subjects**: `subject_id`
- students** to **classes**: `class_id`
- teachers** to **subjects**: `subject_id`







```
1 Base_Metagraph_Traversal(DB, SV)
2   queue  $\leftarrow \emptyset$ 
3   visited  $\leftarrow \emptyset$ 
4   for each v in SV
5     do Enqueue(queue, v)
6   while queue  $\neq \emptyset$ 
7     do cur_v  $\leftarrow$  Dequeue(queue)
8       Add(visited, cur_v)
9       for each u in Adjacent(DB, cur_v)
10         do if u  $\notin$  visited
11           then Enqueue(queue, u)
12   return visited
```



**Предпосылки** к использованию правил метаграфа:

- изменение связей в метаграфе;
- ограничение обхода метаграфа.

**Определение:**  $r = \langle p, f \rangle$  – правило метаграфа, где  $p$  – предикат, а  $f$  – функция, изменяющая множество рёбер (метарёбер) метаграфа.

$p : e \Rightarrow \{True, False\}$ , где  $e \in E$ ,  $E$  – множество рёбер (метарёбер).

$f : E \Rightarrow E'$ , где  $E$  и  $E'$  – множество и обновлённое множество рёбер (метарёбер).



```
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_incident_edges)
15      for each u in Adjacent(MG, cur_v)
16        do if u ∉ visited
17          then Enqueue(queue, u)
18   return visited
```



Разработан язык на основе грамматики SQL стандарта SQL-92 с помощью инструмента ANTLR4.



```
<...>
graph_source_stmt: GRAPH_ SOURCE_ table_name (WHERE_ expr)?;
include_edge_stmt: INCLUDE_ EDGE_ table_name DOT column_name
    table_name DOT column_name;
exclude_edge_stmt: EXCLUDE_ EDGE_ table_name DOT column_name
    table_name DOT column_name;
no_enter_stmt: NO_ ENTER_ table_name (WHERE_ expr)?;
no_exit_stmt: NO_ EXIT_ table_name (WHERE_ expr)?;
limit_visits_stmt: LIMIT_ VISITS_ INTEGER_LITERAL FOR_
    table_name;
limit_distance_stmt: LIMIT_ DISTANCE_ INTEGER_LITERAL FOR_
    table_name;
transformer_stmt: TRANSFORMER_ function_call FOR_ table_name
    DOT column_name (COMMA table_name DOT column_name)*;
<...>
```



```
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;
```

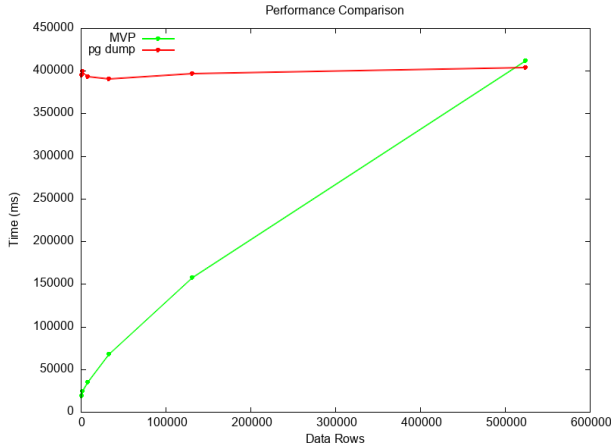


- интерфейс командной строки;
- перенос взаимосвязанных данных;
- построение метаграфа по базе данных;
- поддержка правил для ограничения обхода метаграфа.





Тестовая база данных: 4.5 ГБ, 44739072 записей



## Результаты:

- спроектирована общая архитектура;
- разработан язык описания обхода и трансформации данных;
- реализован минимально жизнеспособный продукт.

## Дальнейшие перспективы:

- улучшение производительности;
- реализация механизмов анонимизации и генерации данных.



Ссылки на код грамматики и код минимально жизнеспособного продукта

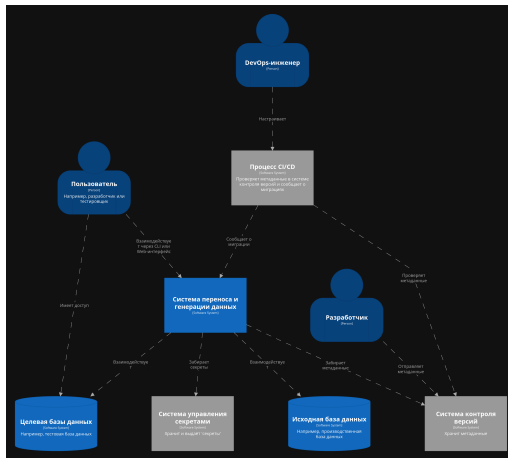


## Алгоритм генерации данных

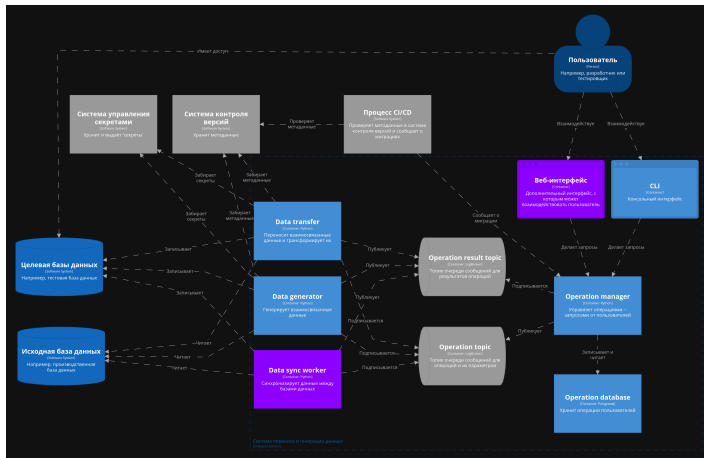
```
1 Data_Generator(MG, RME, SMV, GC, AV)
2   MG.ME  $\leftarrow$  Apply_Rules(MG.ME, RME)
3   vertexes  $\leftarrow \emptyset$ 
4   for each mv in SMV
5     do queue  $\leftarrow$  {mv}
6     visited  $\leftarrow \emptyset$ 
7     while queue  $\neq \emptyset$ 
8       do cur_mv  $\leftarrow$  Dequeue(queue)
9         Add(visited, cur_mv)
10        Extend(vertexes, Gen(cur_mv, GC, AV))
11        for each mu in Adjacent(MG, cur_mv)
12          do if mu  $\notin$  visited
13            then Enqueue(queue, mu)
14   return vertexes
```



## Архитектура на уровне контекста



## Архитектура на уровне контейнеров



## Архитектура на уровне компонент (Data Transfer)

