Санкт-Петербургский государственный университет
Кафедра информационно-аналитических систем

Разработка нового представления закономерностей в проекте Desbordante

Шлёнских Алексей Анатольевич, группа 25.М41-мм

Научный руководитель: Г. А. Чернышев, ассистент кафедры ИАС

Санкт-Петербург
2026

Введение

- науки о данных;
- закономерности;
- поиск — NP-трудная задача, экспоненциальный рост размера результата;
- эталонные реализации на медленных языках.

Desbordnate:

- проект начинался как консольное приложение;
- реализация алгоритмов на C++;
- алгоритмы получили некоторое ускорение;
- стал Python библиотекой;
- другой способ взаимодействия — старая архитектура доставляет проблемы;
- новая задача — проверка соблюдения;
- процесс разработки далёк от идеала — копирование подходов.

Проблем много, исправлять долго, проект существовал и расширялся с существующей архитектурой несколько лет.

Использование

Раньше:

```
$ ./Desbordante_run --algo SomeAlgorithm --param1 val1 --param2 val2
[Col1 Col2] -> Col3
[Col4 Col5] -> Col6
[Col8] -> Col10
...
```

Теперь:

```
def get_results(*args, **kwargs):
    algo = # создание и запуск
    return algo.get_results()
    # algo удалён сборщиком мусора

pickle.dump(result, file=file)

fd_verifier.execute(lhs_indices=[0], rhs_indices=[2])
```

Постановка задачи

Целью работы является обзор текущих проблем, связанных со способом представления закономерностей в коде, и разработка подхода к их исправлению. Для её выполнения были поставлены следующие задачи:

1. Выполнить анализ проблем проекта, возникающих из-за способа представления зависимостей.
2. Предложить вариант архитектуры, который их избегает.
3. Экспериментально его реализовать.

Экспериментальная реализация была проведена для закономерности, называемой функциональной зависимостью (Functional Dependency).

Функциональная зависимость

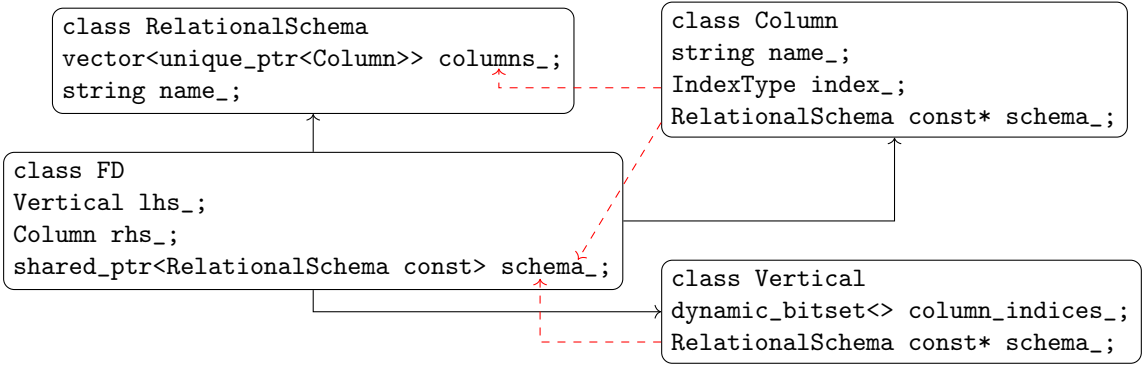
Определяется для некоторой таблицы. Функциональная зависимость $X \to Y$ соблюдается, если по значениям атрибутов из X (называем их левой частью) можно точно определить значение атрибутов из Y (называем их правой частью).
Формально:

$$\forall (p, q) \in (r \times r)(\forall A \in X \;\; p[A] = q[A] \implies \forall B \in Y \;\; p[B] = q[B])$$

Задача поиска ФЗ на некоторой таблице — найти некоторый репрезентативный набор соблюдающихся.
Для задачи поиска допустимо представление с одним атрибутом справа:

$$\forall (p, q) \in (r \times r)(\forall A \in X \;\; p[A] = q[A] \implies p[Y] = q[Y])$$

Представление FD



- сложная сериализация;
- сложно реализовать __eq__ и __hash__;
- не прослеживается время жизни;
- тяжело создавать.

Проверка соблюдения

```
# прочитать данные
data = pd.read_csv(table, header=[0])
# создать алгоритм и загрузить данные
algo = desbordante.afd_verification.algorithms.Default()
algo.load_data(table=data)
# выполнить проверку соблюдения FD из столбца с индексом 0
# в столбец с индексом 2
algo.execute(lhs_indices=[0], rhs_indices=[2]) # !!
```

- нет никакой связи с классом FD;
- нельзя указывать названия столбцов вместо индексов, неудобно.

Вид решения

Имеем классы нескольких типов:

- внутренние;
- хранение;
- представление;
- ввод.

Внутренние классы — обеспечение быстрой работы алгоритма. Зависят от алгоритмов, обычно являются чем-то простым (числа, битовые маски).

Представление — обеспечение удобства работы пользователя. Каждый тип закономерности представлен одним набором классов, самостоятельное время жизни, большие по размеру (строки, массивы).

Хранение — переходное состояние, обеспечивают минимизацию потребления памяти и быструю запись ответа.

Ввод — обеспечение удобства указания закономерности для проверки.

Представление

```
struct Attribute {
    std::string name;
    Index id;
};
struct FunctionalDependency {
    std::string table_name;
    std::vector<Attribute> lhs;
    std::vector<Attribute> rhs;
    /* методы */
};
```

- нет рекурсивности;
- простые элементы класса;
- легко реализуются методы;
- `id` нужно, так как возможны одинаковые названия столбцов.

Хранение

```
struct TableHeader {
    std::string table_name;
    std::vector<std::string> column_names;
};
struct StrippedFd {
    boost::dynamic_bitset<> lhs;
    boost::dynamic_bitset<> rhs;
};
class FdStorage {
    TableHeader table_header_;
    Container<StrippedFd> fds_;
};
```

- алгоритму достаточно скопировать битовую маску;
- доступ с помощью итерации (`std::ranges::views::transform`).

Ввод

```
struct FdInput {
    std::vector<std::variant<std::string, Index>> lhs;
    std::vector<std::variant<std::string, Index>> rhs;
};
```

Предполагается использование названия столбца, но в редком случае, когда столбцов несколько, можно использовать и индекс для устранения двусмысленности. Зависимость подаётся на вход следующим образом:

```
fd_verifier.execute(fd=FdInput(["id"], ["name"]))
```

Также классу представления функциональной зависимости добавлен метод преобразования к такому виду:

```
fd_verifier.execute(fd=fd.to_input())
```

Заключение

В ходе работы были проработаны проблемы проекта Desbordante, связанные со способом представления закономерностей.

- Был выполнен анализ требований к проекту и проблем с текущим представлением закономерностей.
- Были разработаны принципы устройства представления закономерностей.
- В качестве эксперимента был реализован подход, использующий эти принципы.