МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

НИЖЕГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ

УНИВЕРСИТЕТ им. Р.Е.АЛЕКСЕЕВА

Институт радиоэлектроники и информационных технологий

Кафедра вычислительные системы и технологии

Пояснительная записка

к курсовой работе

по дисциплине «Принципы и методы организации системных программных средств»

на тему: «Исследование алгоритмов построения сетевых траекторий обучения на основе распределенной базы данных»

РУКОВОДИТЕЛЬ:

_____ Кочешков А.А.

СТУДЕНТ:

_____ Сапожников В.О.

19-В-1

Работа защищена «___» _____

С оценкой _____

Нижний Новгород 2022

Содержание

1. Введение

При проектировании и разработке образовательных траекторий учебного процесса важнейшими вопросами являются обеспечение эффективности процесса подготовки и оптимизация затрат по времени ресурсам на формировании компетенции выпускников. Определение количества учебных курсов, объединение их в последовательности, в которых предыдущие курсы формируют компетенции, необходимые для освоения последующих, невозможно без структуризации и стандартизации компетенций.

В связи с этим можно констатировать факт отсутствия единых информационных пространств и коммуникационных сред, которые бы обеспечивали хранение и распределенную обработку спецификаций компетенций и учебных курсов, реализующих цепочку преобразований компетенций в машиночитаемом формате.

Особенную актуальность обозначенные вопросы приобрели в контексте глобализации, интеграции сферы высшего образования, необходимости кооперации ВУЗов в рамках сетевых образовательных программ.

Процесс формирования учебных планов для образовательных программ высшего образования связан с необходимостью учитывать ряд факторов, прямо или косвенно связанных с организацией учебного процесса.

Для направлений подготовки 09.03.01 и 09.04.01 «Информатика и вычислительная техника» задача формирования учебного плана дополнительно усложняется, поскольку появление новых и развитие существующих технологий в области IT приводит к необходимости регулярной и своевременной актуализации учебных планов путем:

- добавления новых учебных курсов;
- замены/удаления устаревших учебных курсов;
- передачи отдельных учебных курсов для реализации другим структурным подразделениям или образовательным учреждениям (формирование сетевых образовательных программ) при наличии у них необходимого материально-технического обеспечения или кадрового ресурса.

В данной работе предложен алгоритм формирования учебных траекторий на основе распределенной базы данных, которая представляет собой общее хранилище курсов и входных и выходных компетенции, реализующихся разными ВУЗами. Для упрощения и гибкости системы компетенции были разбиты на составляющие, представляющие собой знания, умения, навыки (ЗУНы).

При выполнении работы считается, что ЗУНы являются стандартизированными и курсы во всех ВУЗах строятся на основе данных знаний, умений и навыков.

1.1. Цели и задачи работы

Целью данной работы является разработка алгоритма построения сетевых траекторий обучения на основе распределенной базы данных, а также разработка внешнего API для выполнения CRUD операций над данными и взаимодействия с интерфейсом.

Данная цель разбивается на следующий ряд задач:

1. Разработка модели курсов и ЗУНов в виде набора данных с предопределенными связями между ними.
2. Реализация распределенной базы данных.
3. Перенос разработанной модели в распределённую базу данных.
4. Реализация внешнего API для выполнения CRUD операций.
5. Разработка алгоритма построения сетевых траекторий обучения на основе распределенной базы данных.
6. Реализация и оптимизация разработанного алгоритма.
7. Разработка внешнего API для взаимодействия с реализованным алгоритмом.

2. Разработка модели курсов и ЗУНов

Для представления курсов и ЗУНов была построена следующая реляционная модель (Рис. 1)

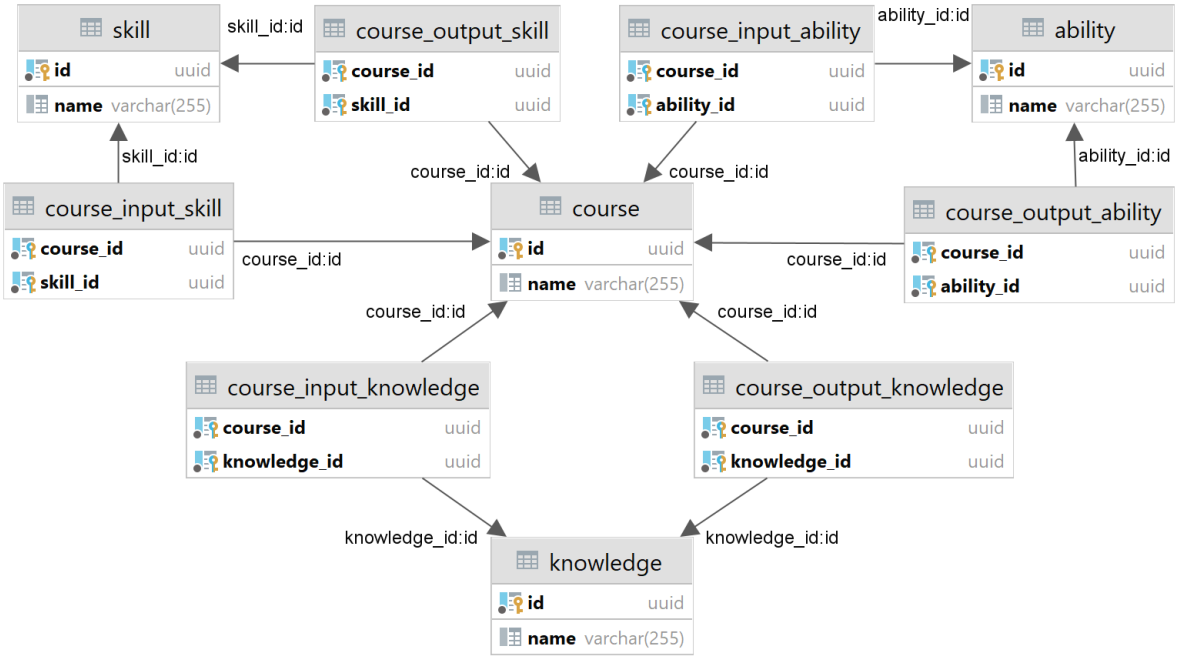

Рисунок 1. - Реляционная модель БД

Справочник ЗУНов представляет собой совокупность словарей SKILL, ABILITY, KNOWLEDGE которые содержат название и первичный ключ.

Курс является более сложной сущностью, обладающей входными и выходными ЗУНами. Для связи ЗУНОВ с курсами предусмотрены следующие таблицы:

- COURSE_INPUT_KNOWLEDGE, COURSE_OUTPUT_KNOWLEDGE - входные и выходные знания;
- COURSE_INPUT_ABILITY, COURSE_OUTPUT_ABILITY - выходные и выходные умения;
- COURSE_INPUT_SKILL, COURSE_OUTPUT_SKILL - входные и выходные навыки;

3. Реализация распределённой базы данных

В качестве системы управления базами данных (СУБД) PostgreSQL.

PostgreSQL разрабатывается сообществом PostgreSQL, имеет лицензию PostgreSQL License и является бесплатно распространяемым программным обеспечением.

3.1. Анализ существующих решений

Для создания распределённой БД на основе PostgreSQL существует несколько способов.

3.1.1. Postgres-XL

Postgres-XL это программное обеспечение системы управления распределенными базами данных, основанное на PostgreSQL. Данное решение хорошо подходит как для обработки транзакций реального времени, так и для создания крупных баз для анализа больших наборов данных. Postgres-XL свободно распространяет под лицензией Mozilla Public License 2.0.

Достоинства Postgres-XL:

1. Соответствие требованиям ACID (атомарность, согласованность, изолированность, надежность) на уровне всего кластера.
2. Поддержка механизма многоверсионности для обеспечения одновременного конкурентного доступа к БД (MVCC).
3. Распределённая модель хранения, при которой каждый узел хранит и обрабатывает отдельную порцию данных. При записи данные равномерно распределяются по разным узлам хранения, что позволяет более эффективно использовать кэширование и распределять нагрузку при чтении.
4. Расширенная модель разграничения доступа, позволяющая организовать в рамках одного кластера несколько виртуальных СУБД, закреплённых за разными арендаторами (Multi-tenant).
5. Поддержка большинства штатных возможностей PostgreSQL, в том числе средств работы с данными в формате JSON и hstore.
6. Высокая масштабируемость - при необходимости наращивания размера базы или при увеличении нагрузки достаточно подключить новые узлы в кластер.
7. Оптимизация кластера как для приложений с большой интенсивностью записи, так и для программ, в которых преобладают операции чтения.
8. Средства обеспечения отказоустойчивости через развёртывание запасных узлов, которые примут нагрузку в случае выхода из строя основного узла.

Так же Postgres-XL поддерживается только на UNIX подобных ОС.

3.1.2. DBlink

DBLink – пришедшее из СУБД Oracle расширение для создания связи с таблицами из удаленных СУБД. На данный момент DBLink поддерживает

связь между большинством крупных СУБД, таких как PostgreSQL, Oracle DB, MS SQL.

Однако у данного механизма есть ряд недостатков:

1. Явное указание параметров подключения при выполнении запросов к удаленным БД.
2. Данный механизм поддерживает только базовые команды для БД, исключая возможность использования специфических операции разных СУБД.

3.1.2. Postgres_fdw

Модуль postgres_fdw предоставляет обёртку сторонних данных postgres_fdw, используя которую можно обращаться к данным, находящимся на внешних серверах PostgreSQL.

Функциональность этого модуля во многом пересекается с функциональностью старого модуля DBLink. Однако postgres_fdw предоставляет более прозрачный и стандартизированный синтаксис для обращения к удалённым таблицам и во многих случаях даёт лучшую производительность.

Так же поскольку модуль postgres_fdw является расширением к СУБД PostgreSQL он позволяет использовать специфичные команды и процедурное расширение языка PL/pgSQL.

Пускай DBLink и Postgres_fdw не являются специализированным ПО для построение распределённой БД они могут предоставить интерфейс для доступа к нескольким БД, тем самым эмулируя распределённую систему.

3.2. Выбор и обоснования решения.

При выполнении данной работы используется расширение Postgres_fdw для СУБД PostgreSQL. Данное решение является более удобным и производительным по сравнению с DBLink, менее сложным в конфигурации и администрировании и менее требовательным к аппаратной части, чем Postgres-XL.

3.3. Создание распределенной базы данных и перенос разработанной модели

3.3.1. Создание баз данных

Для создания распределённой БД используется СУБД PostgreSQL 14.2-2 для 64 битной Windows 10.

Используемая при выполнении данной работы распределенная система включает в себя три базы данных курсов и ЗУНов (ЗУНы в каждой БД одинаковы): NNTU, NNGU, MTUCI.

Одной из основной идей данной системы является то, что мы можем задействовать курсы разных университетов.

В каждой из БД воссоздаем разработанную модель (Рис. 1), при этом все названия таблиц в каждой БД одинаковы.

Таблицы курсов и ЗУНов представляют собой словари, состоящие из имени и идентификатора.

Скрипт для создания таблиц курсов и знаний:

```
CREATE TABLE KNOWLEDGE (
    id UUID NOT NULL UNIQUE PRIMARY KEY,
    name VARCHAR(255)
);
CREATE TABLE SKILL (
    id UUID NOT NULL UNIQUE PRIMARY KEY,
    name VARCHAR(255)
);
```

Остальные таблицы ЗУНов создаются аналогичным способом.

Связь между курсами и наборами входных и выходных ЗУНов осуществляется при помощи отношения многие-ко-многим, поскольку кол-во входных и выходных знаний, умений и навыков у каждого курса может быть разным.

Скрипт создания таблицы связи курсов и входных знаний:

```
CREATE TABLE COURSE_INPUT_KNOWLEDGE (
    course_id UUID NOT NULL,
    knowledge_id UUID NOT NULL,

    CONSTRAINT course_input_knowledge_pk
        PRIMARY KEY (course_id, knowledge_id),
    CONSTRAINT course_input_knowledge_fk
        FOREIGN KEY (course_id)
        REFERENCES COURSE(id)
        ON DELETE CASCADE,
    CONSTRAINT input_knowledge_fk
        FOREIGN KEY (knowledge_id)
        REFERENCES KNOWLEDGE(id)
        ON DELETE CASCADE
);
```

Остальные таблицы связи создаются аналогичным способом.

3.3.2. Создание единого интерфейса

Распределённая база данных подразумевает, что мы имеем один интерфейс для доступа ко всем базам данных.

Для начала нам необходимо выбрать “главный узел” нашей распределенной из которой мы будем обращаться к остальным базам. В нашем случае это будет база NGTU.

Затем на выбранный узел необходимо установить расширение postgres_fdw:

```
CREATE EXTENSION postgres_fdw;
```

Информация об установленных расширениях хранится в таблице pg_extension:

```
SELECT * FROM pg_extension;
```

	oid	extname	extowner	extnamespace	extrelocatable	extversion
1	13740	plpgsql	10	11	false	1.0
2	17282	postgres_fdw	10	2200	true	1.1

Затем для каждой удаленной базы данных необходимо создать сторонний сервер.

Создание сервера для подключения к БД MTUCI:

```
CREATE SERVER mtuci_server
    FOREIGN DATA WRAPPER postgres_fdw
    OPTIONS (host 'localhost', port '5432', dbname 'mtuci');
```

Для определения роли, которая будет задействована на удаленном сервера задаётся сопоставление пользователей. Для упрощения работы для работы в каждой из БД используются данные суперпользователя postgres:

```
CREATE USER MAPPING FOR postgre
        SERVER mtuci_server
        OPTIONS (user 'postgre', password 'postgre');
```

Информация об удаленных серверах и сопоставлениях ролей хранятся в таблицах pg_foreign_server и pg_user_mapping соответственно:

```
SELECT * FROM pg_foreign_server;
```

	oid	srvname	...	srvfdw	...	...	...	srvoptions
1	20684	mtuci_server	10	20678	<...	<nu...	<n...	{host=localhost,port=5432,dbname=mtuci}
2	20716	nngu_server	10	20678	<...	<nu...	<n...	{host=localhost,port=5432,dbname=nngu}

```
SELECT * FROM pg_user_mapping;
```

	oid	umuser	umserver	umoptions
1	20685	10	20684	{user=postgres,password=postgre}
2	20717	10	20716	{user=postgres,password=postgre}

Расширения, удаленные сервера и сопоставления ролей являются объектами базы данных, внутри которых они созданы.

Поскольку при работе будут использоваться все таблицы из всех баз данных, то самым простым способом создания сторонних таблиц будет импортирование всех таблиц определённой схемы:

```
IMPORT FOREIGN SCHEMA courses
    FROM SERVER mtuci_server INTO mtuci_courses;
```

Из схемы courses из БД MTUCI импортируем все таблицы в локальную схему mtuci_courses. Таким образом мы получили доступ к таблицам БД MTUCI из БД NGTU (Рис. 3)

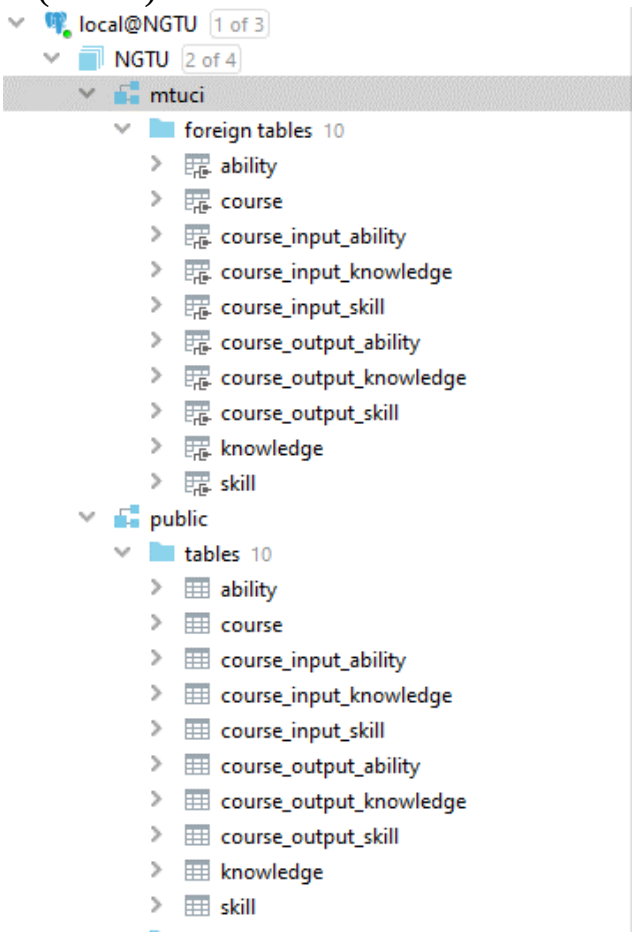

Рисунок 2. - Доступ к укадленным таблицам

Повторив данные действия для каждой удаленной БД, мы получим единый интерфейс для взаимодействия с таблицами разных БД на главном узле NGTU.

3.4. Настройка удаленного доступа к серверу Postgres

Разработка велась с двух машин: стационарный персональный компьютер и ноутбук. Поскольку используется локальная база данных PostgreSQL, то возникает проблема переноса изменений между базами данных на разных машинах.

Для решения данной проблемы был настроен удаленный доступ к серверу PostgreSQL на ПК.

Сначала необходимо создать пользователя базы данных, который будет использован для удаленной работы и настроить права пользователя для работы с базой данных. Для упрощения был использован профиль суперпользователя Postgres, который обладает максимальными правами. В данном случае не рассматривается вопрос безопасности, поскольку с БД работает один и тот же человек.

По умолчанию PostgreSQL на Windows x64 устанавливается по пути: C:\Program Files\PostgreSQL\<version>

Переходим в каталог data и в текстовым редакторе открываем файл pg_hba.conf(Рис. 3)

```
# TYPE  DATABASE        USER            ADDRESS                 METHOD

# "local" is for Unix domain socket connections only
local   all             all                                     scram-sha-256
# IPv4 local connections:
host    all             all             127.0.0.1/32            scram-sha-256
# IPv6 local connections:
host    all             all             ::1/128                 scram-sha-256
# Allow replication connections from localhost, by a user with the
# replication privilege.
local   replication     all                                     scram-sha-256
host    replication     all             127.0.0.1/32            scram-sha-256
host    replication     all             ::1/128                 scram-sha-256
```

Рисунок 3. Содержание pg_hba.conf

Файл имеет следующую структуру (Таблица 1):

Вид подключения	Наименование базы данных	Имя пользователя	IPадрес удаленного рабочего места	Метод аутентификации

Таблица 1. - Структура pg_hba.conf

В виде подключения указываем: host – используется подключение по TCP/IP

В наименовании базы данных указываем имена БД, к которым мы хотим предоставить доступ. Т.к. мы используем три БД, то придётся продублировать строки подключения для каждой из них.

В имени пользователя указываем имя профиля, под которым будет подключение

Указываем IP адрес удаленного рабочего места. В качестве безопасности было решено задать конкретный адрес IPv4. Для неизменяемости IP адреса подключаемого устройства было решено связать хосты при помощи VPN построенной специализированным ПО - LogMeInHamachi. Внутри виртуальной сети машины имеют статические IP-адреса.

Метод аутентификации scram-sha-256 стандартный для PostgreSQL последних версий.

Таким образом итоговый файл имеет вид (Рис. 4):

```
# TYPE  DATABASE        USER            ADDRESS                 METHOD

# "local" is for Unix domain socket connections only
local   all             all                                     scram-sha-256
# IPv4 local connections:
host    all             all             127.0.0.1/32            scram-sha-256
# IPv6 local connections:
host    all             all             ::1/128                 scram-sha-256
# Allow replication connections from localhost, by a user with the
# replication privilege.
local   replication     all                                     scram-sha-256
host    replication     all             127.0.0.1/32            scram-sha-256
host    replication     all             ::1/128                 scram-sha-256
host    nntu            postgres        25.17.151.180/32        scram-sha-256
host    ngtu            postgres        25.17.151.180/32        scram-sha-256
host    mtuci           postgres        25.17.151.180/32        scram-sha-256
```

Рисунок 4. - Измененный pg_hba.conf

Теперь необходимо перезапустить сервер PostgreSQL для обновления конфигурации. Для этого переходим в Панель управления → Администрирование → службы и находим там postgresql-x64-14-PostgreSQL Server 14 (Рис. 5)

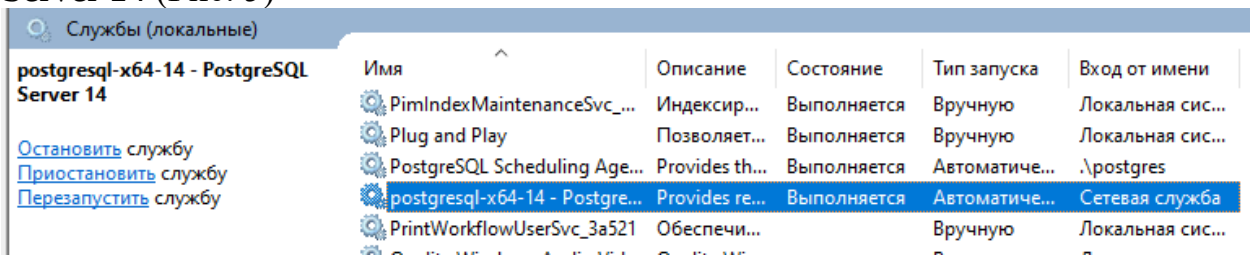

Рисунок 5. - Служба postgresql-x64-14

Затем необходимо открыть порт для подключения. Для PostgreSQL стандартным портом является 5432. Для этого переходи в Панель управления → Брандмауэр защитника Windows → Дополнительные параметры → Правила для входящий подключений → Создать правило (Рис. 6)

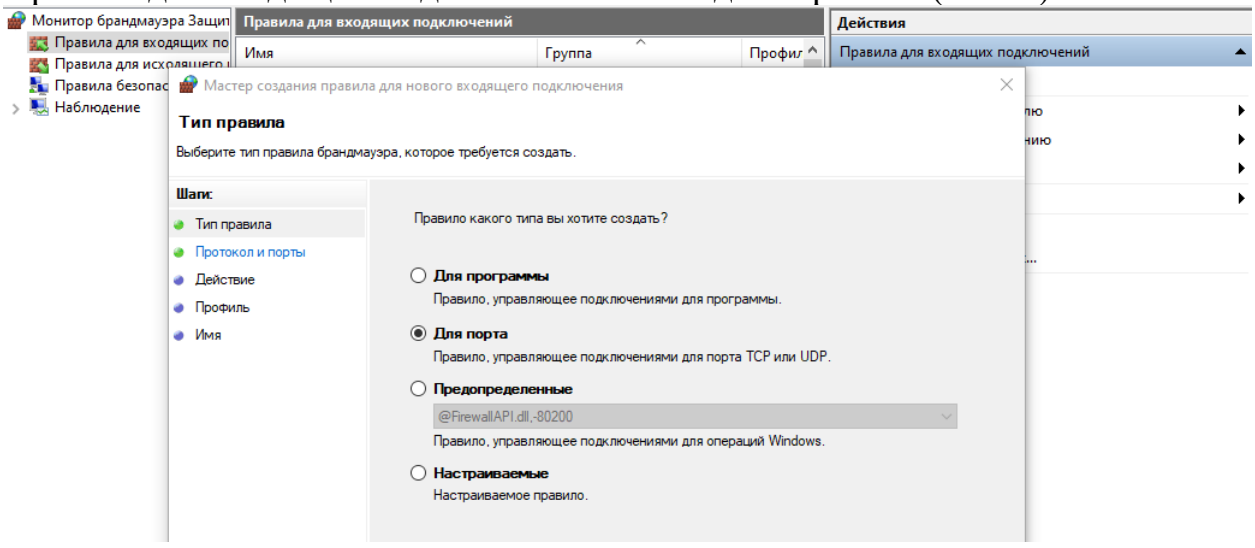

Рисунок 6. - Создание правила для порта

Выбираем создать правило для порта, указываем что подключение будет по протоколу TCP и указываем номер локального порта (Рис.7)

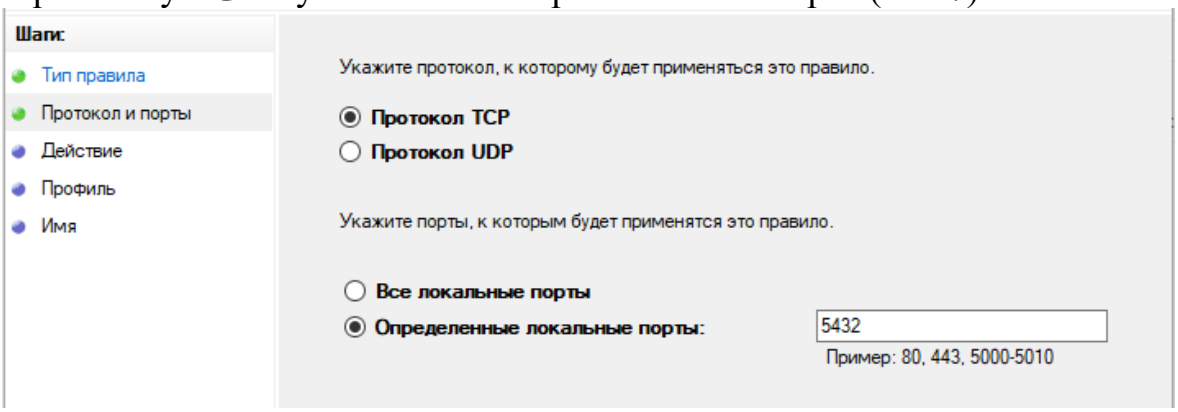

Рисунок 7. - Выбор протокола и порта

Затем мы указываем какое действие выполняться при удовлетворении указанным ранее условиям, в нашем случае мы разрешаем подключение (Рис. 8):

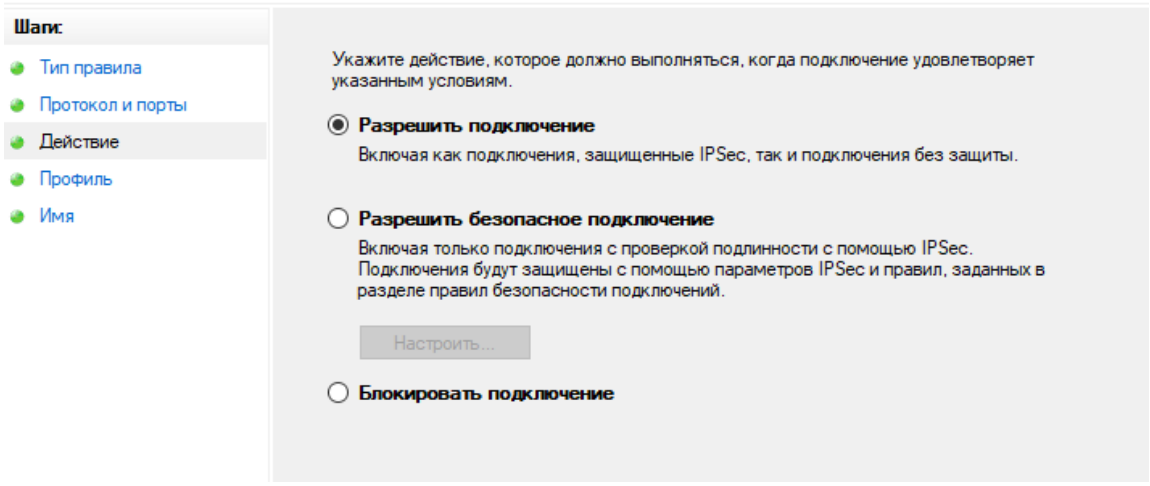

Рисунок 8. - Разрешение подключения

Указываем для какого профиля сетевого подключения используется данное правило (Рис.9):

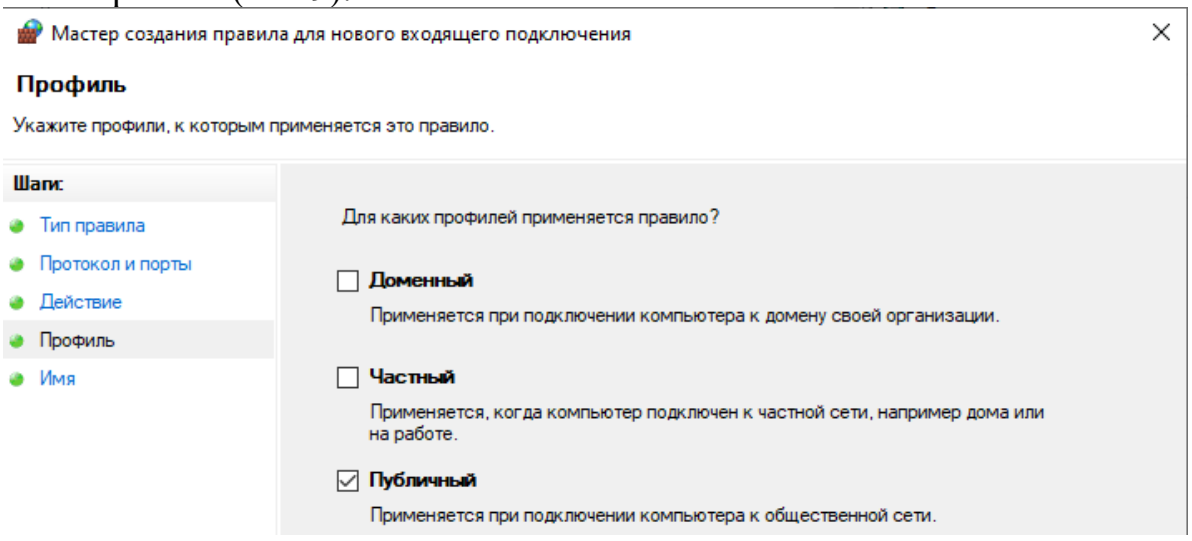

Рисунок 9. - Выбор сетевого профиля

Задаем имя для указанного правила (Рис.10):

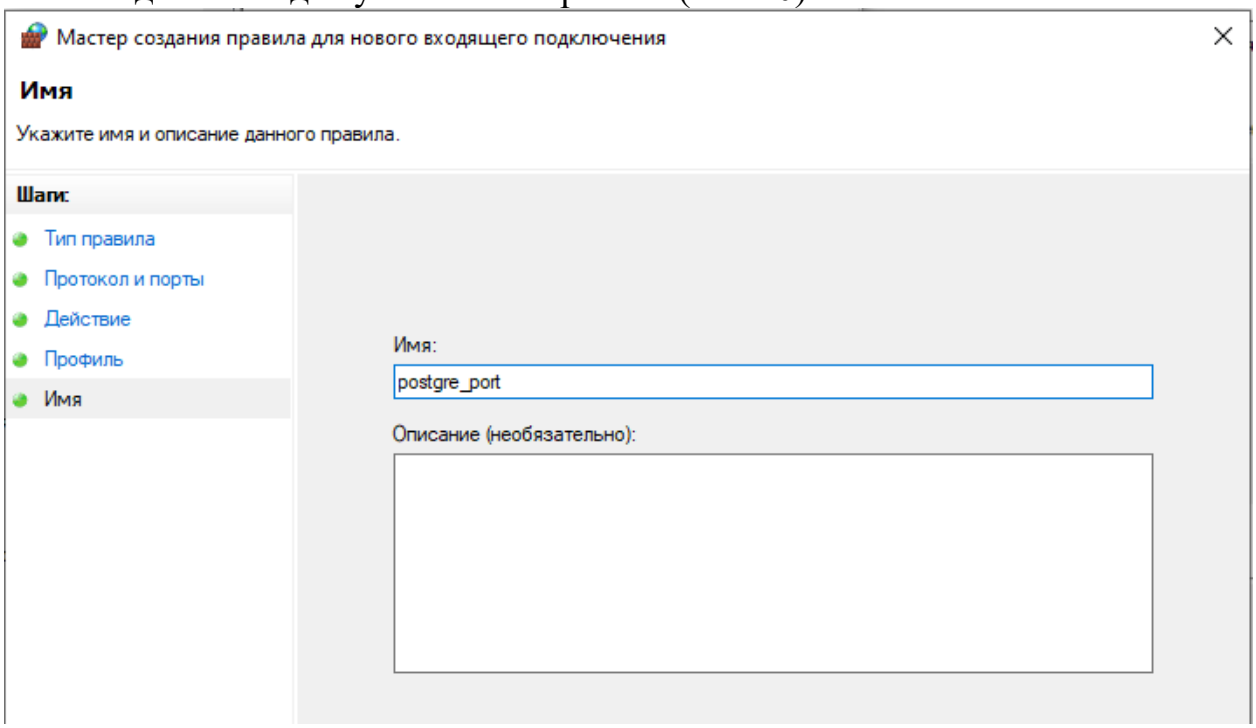

Рисунок 10. - Задание имени правилу

После выполнения всех указанных действий наше правило вступает в силу (Рис.11).

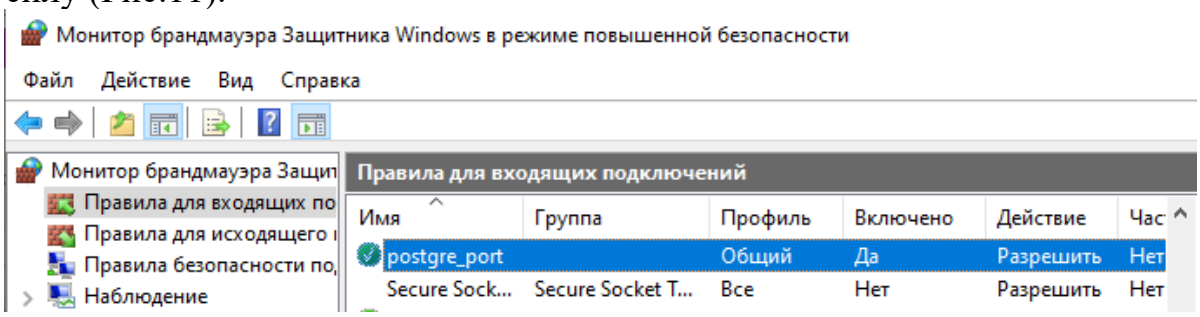

Рисунок 11. - Задание имени правилу

Данный способ установки удаленного подключения является не совсем правильным, т.к. указанный порт открыт для любых TCP подключений, но, с другой стороны, он не слушает никакими способами кроме PostgreSQL.

4. Создание API для выполнения CRUD операций

Для работы с данными базами был реализован API на ЯП Scala.

Scala — мультипарадигмальный язык программирования, спроектированный кратким и типобезопасным для простого и быстрого создания компонентного программного обеспечения, сочетающий возможности функционального и объектно-ориентированного программирования.

Scala имеет реализации на платформах JavaScript и Java Virtual Machine (JVM). При разработке использовалась последняя реализация, что обеспечивает кроссплатформенность.

Для работы с базами данных использовалась библиотека ScalikeJDBC, которая основана на Java DataBase Connectivity (JDBC). JDBC – платформенно независимый промышленный стандарт взаимодействия Java – приложений с различными СУБД.

Библиотека ScalikeJDBC предоставляет несколько способов взаимодействия с БД: при помощи SQL интерполятора и type-safe DSL.

SQL интерполятора позволяет вставлять внутри кода Scala вставлять код SQL, который без обработки будет передан на исполнения БД. Такой подход выигрывает по времени у type-safe DSL, поскольку скрипты SQL не подвергаются предварительной обработке и проверке, но является не безопасным и подвержен SQL инъекциям.

Type-safe DSL позволяет описывать запросы к БД на языке Scala, перед отправкой запроса он проверяется на корректность, переводится в SQL и только затем передается на исполнение. Данный способ обладает ограниченным функционалом и позволяет выполнять лишь базовые CRUD операции.

В ходе выполнения работы был реализован API при помощи обоих способов и проведен сравнительный анализ скорости выполнения запросов выборки (Рис.12) и вставки (Рис.13):

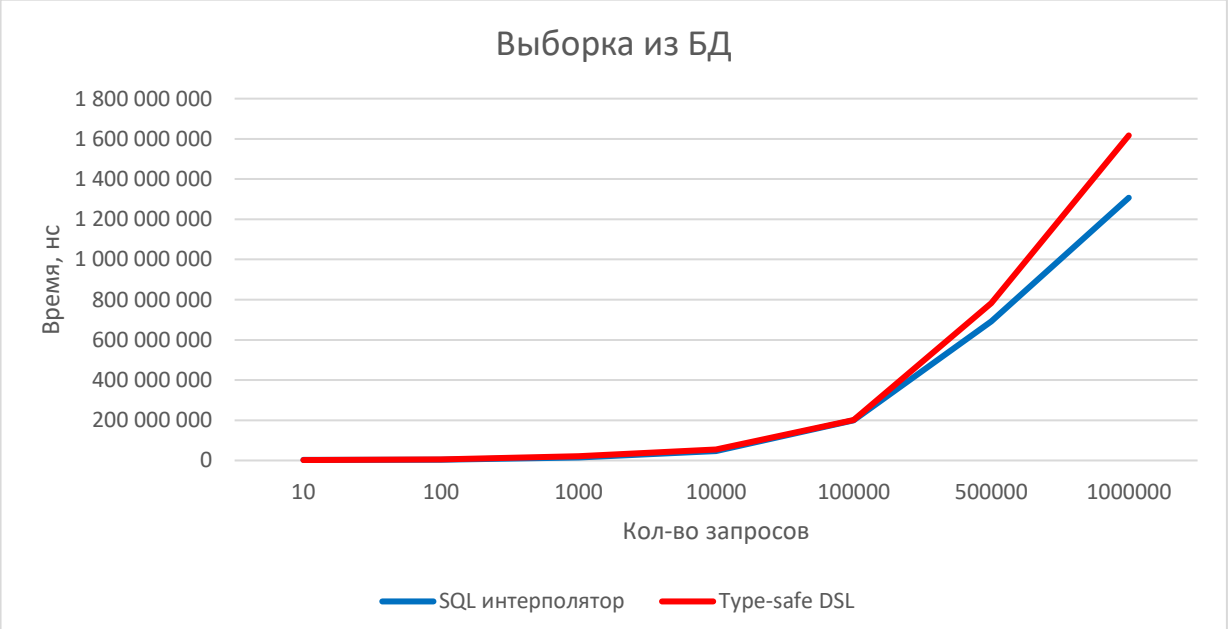

Рисунок 12. - Сравнение выборки при помощи SQL интерполятора и Type-safe DSL

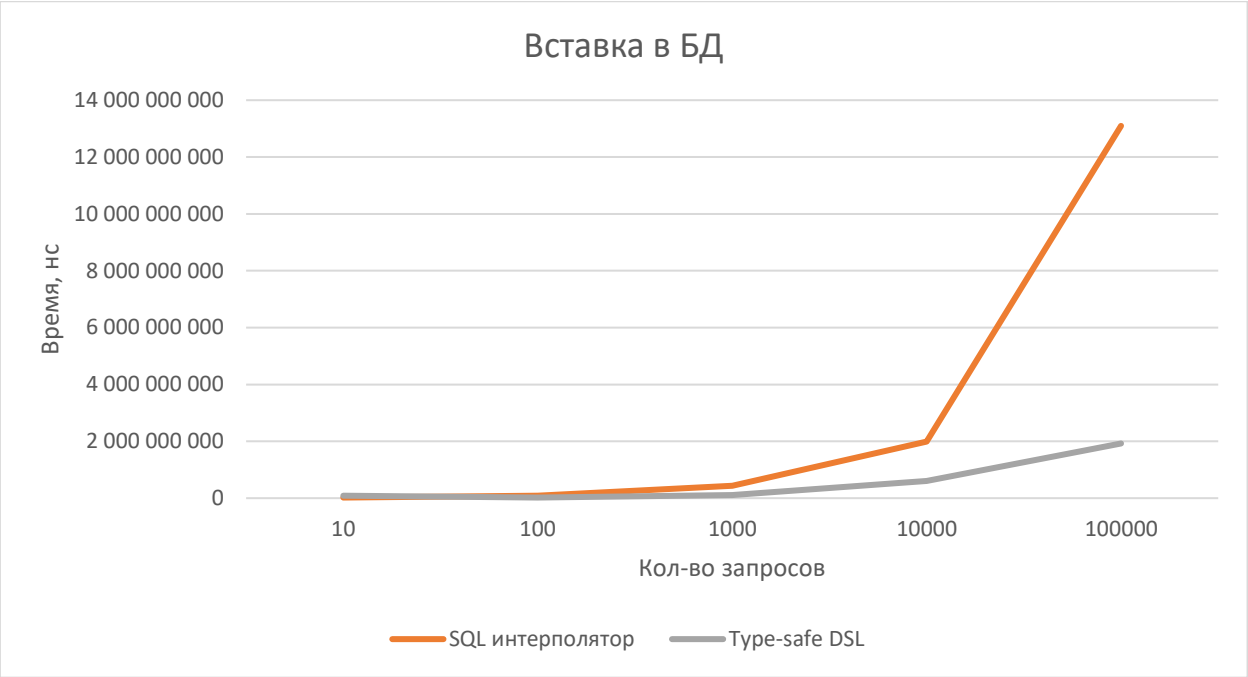

Рисунок 13. - Сравнение вставки при помощи SQL интерполятора и Type-safe DSL

При большом кол-ве запросов на выборку данных Type-safe DSL незначительно уступает использованию SQL-интерполятора, тратя время на внутреннею проверку и обработку запросов перед передачей на исполнение.

При вставке большего кол-ва данных Type-safe DSL значительно выигрывает в скорости из-за оптимизации передачи множества данных на вставку.

В итоге было решено остановится на API, реализованном при помощи Type-safe DSL.

5. Алгоритм построение сетевых учебных траекторий

5.1. Первая версия алгоритма

Для построения образовательных траекторий был разработан следующий алгоритм (Рис.14):

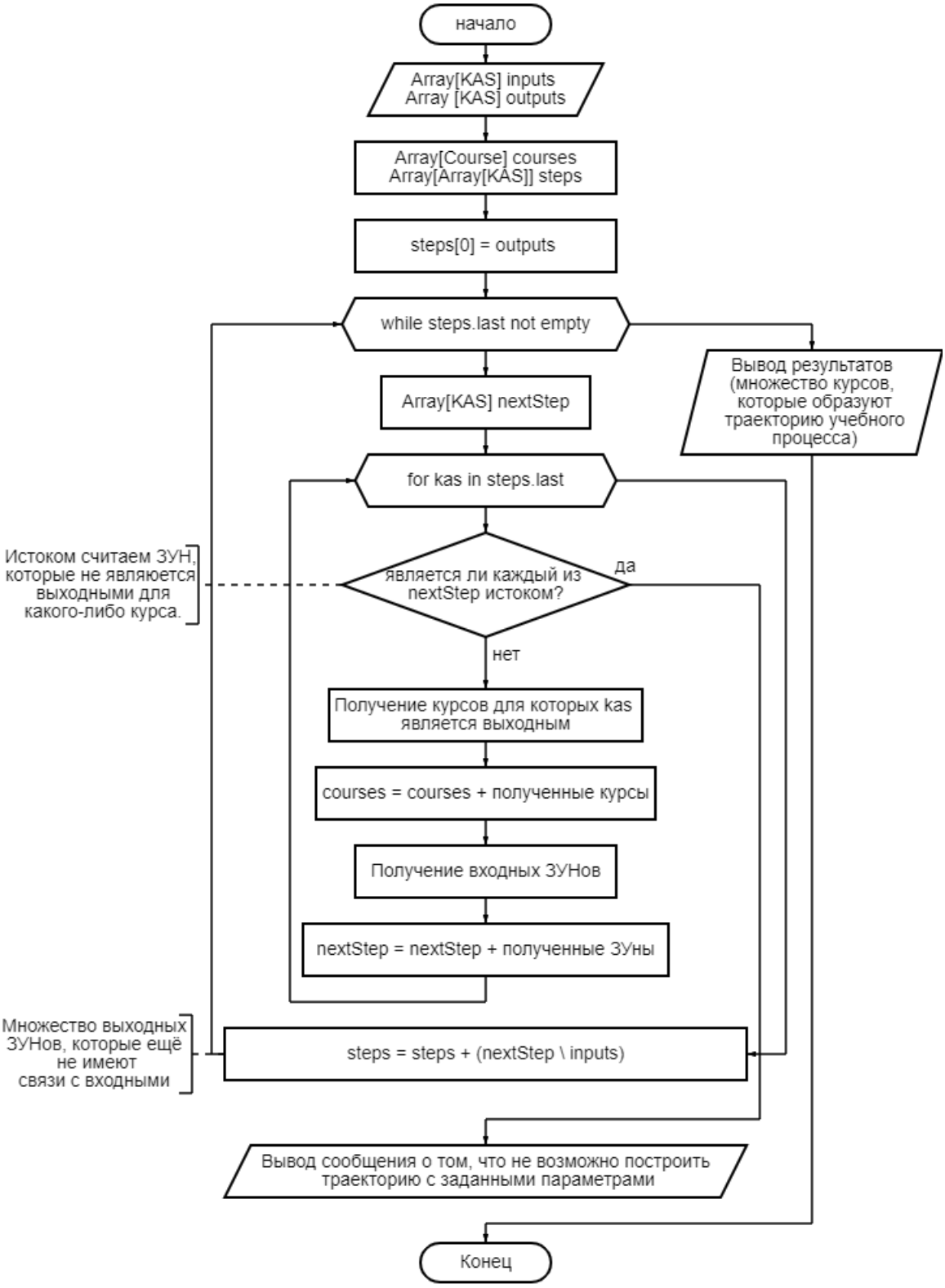

Рисунок 14. Первая версия алгоритма

(Knowledge, Ability, Skill = Знание, Умения, Навыки)

От выходных ЗУНов ведется поиск в ширину, пока для каждого из выходных ЗУНов не будет построен путь хотя бы до одного из входных ЗУНов. Если мы дошли до ЗУНов, которые являются истоками – не являются выходными для какого-либо курса, то алгоритм прекращает работу. т.к. невозможно построить образовательную траекторию по заданным входным и выходным ЗУНам.

5.2. Оптимизация алгоритма

Для оптимизации выполнении алгоритма по времени и памяти было решено, что алгоритм должен выполняться на стороне БД. Таким образом мы избегаем дополнительных затрат по выборке данных из БД и их хранения в момент выполнения. Так же работая на уровне БД мы можем работать не с целыми строками таблиц, представляющими собой объекты курсов и ЗУНов, а только с id интересующих нас данных.

При реализации алгоритма была проведена оптимизация, позволяющая избежать излишнего обращения к таблицам.

Проверка является ли ЗУН (KAS) истоком требует обращения к таблице связи курса и выходного ЗУНа и следующий шаг с получением курсов, для которых ЗУН(KAS) является выходным так же требует обращения к этой же таблице. Было решено объединить эти обращения в одно, а проверку проводить по множеству полученных курсов. Если множество полученных курсов является пустым, то рассматриваемые ЗУНы (KAS) являются истоками.

Алгоритм принял следующий вид (Рис.15):

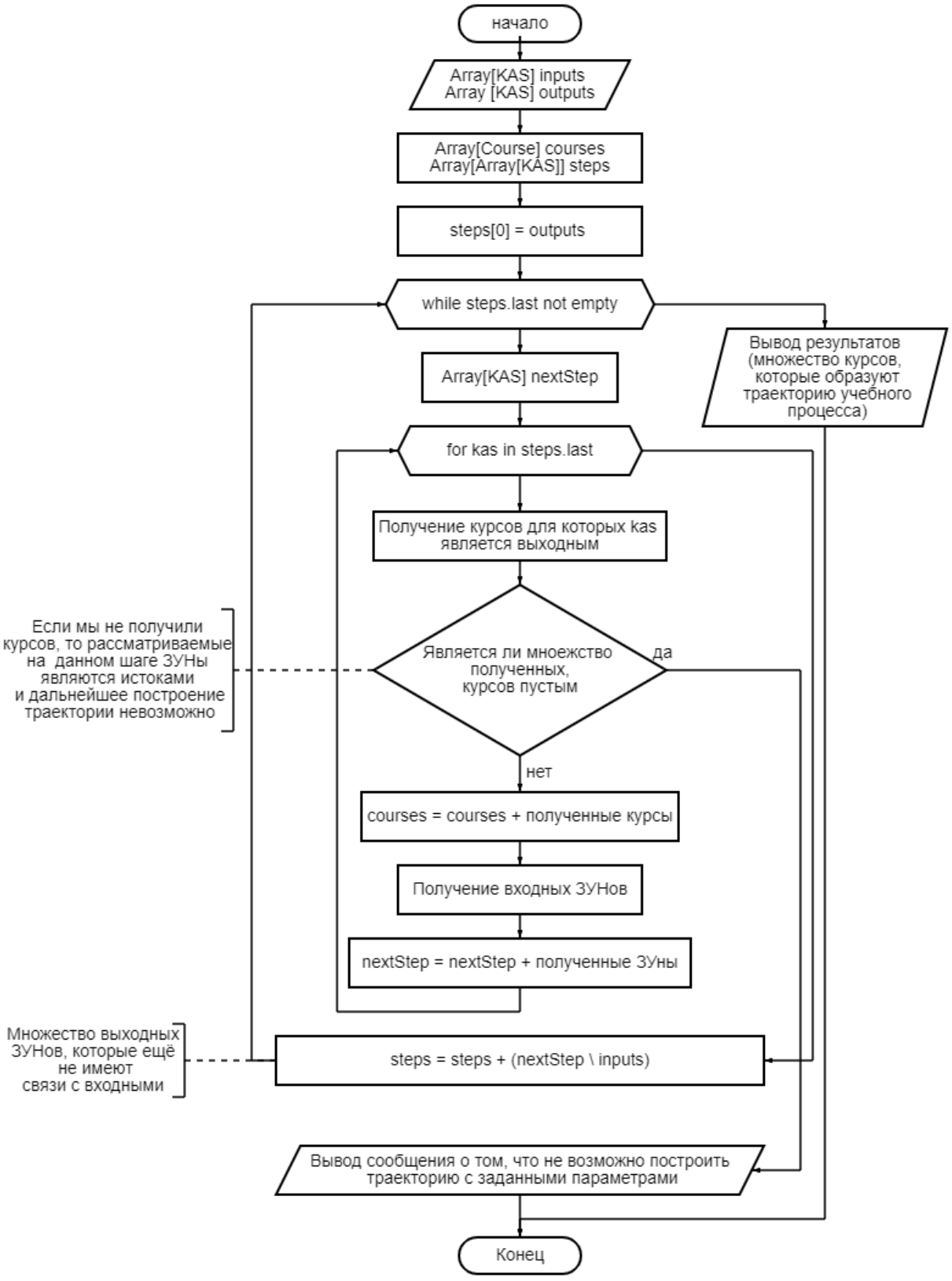

Рисунок 15. - Оптимизированный алгоритм

5.3. Сложности при реализации алгоритма

СУБД PostgreSQL поддерживает три языка: C, SQL и процедурное расширение языка SQL используемое в данной СУБД – PL/pgSQL. Алгоритм реализован на PL/pgSQL.

В СУБД PostgreSQL реализованы возможности создания хранимых процедур, однако они не полностью совместимы с процедурами языка SQL, поэтому рекомендуется использовать хранимые функции.

Обобщённый вид функции на языке PL/pgSQL:

```
CREATE [OR REPLACE] FUNCTION <имя функции> ([входные параметры])
RETURNS <Возвращаемый тип> AS
$метка$
[DECLARE]
    [Объявление локальный переменных]

BEGIN
    <Исполняемый код>

    RETURN [возвращаемое значение]
END;
$метка$ LANGUAGE plpgsql;
```

Язык PL/pgSQL не обладает возможностями и изобилием “синтаксического сахара”, которые присуще современным высокоуровневым ЯП, а также имеет свою специфику, в связи с чем возник ряд сложностей:

1. Операции разности множеств не поддерживается для массивов, но поддерживается для таблиц. Одним из способов нахождения разницы массивов является представление из в виде таблиц, нахождения разницы между таблицами и затем вернуть полученный результат в вид массива. Встроенная операция *unnest(<anyarray>)* позволяет представить массив в виде таблицы. Таким образом для нахождения разности массива была реализована следующая функция:

```
CREATE OR REPLACE FUNCTION arrDif(minuend anyarray,
subtrahend anyarray) RETURNS anyarray AS
$code$
BEGIN
    RETURN (SELECT array
                        (SELECT unnest(minuend)
                         EXCEPT
                         SELECT unnest(subtrahend)));
END;
$code$ LANGUAGE plpgsql;
```

2. Язык PL/pgSQL не поддерживает многомерные массивы. Для эмуляции многомерного массива было решено создать временную таблицу, которая содержит в себе id, обозначающий индекс массива и столбец массивов:

```
CREATE TEMP TABLE steps
(
    stepNumber SERIAL,
    step       UUID[]
);
```

3. В функциях, написанных на PL/pgSQL на стадии компиляции уже должны быть известны имена и сигнатуры таблиц, к которым идёт обращение в коде. Таким образом теряется гибкость функций, мы не можем передать имена таблиц как параметры. Поскольку курсы имеют наборы входных и выходных знаний, умений и навыков, который представляют собой разные таблицы, то все части алгоритма, где требуется обращения к таблицам ЗУНов или таблицам связей курсов и ЗУНов, должны быть продублированные для каждого из видов ЗУНов. Поэтому алгоритм выполняется в трех разных “плоскостях” – по знаниям, умениям и навыкам, затем полученные множества объединяются и из результат отбрасываются повторяющиеся элементы.

4. В языке PL/pgSQL нет встроенных функций проверки массива на пустоту, а самым быстро действенным способом проверки является сравнение строкового представления массива со строкой, обозначающей пустой массив:

```
<имя массива> = '{}'
```

6. Тестирование

6.1. Подготовка к тестированию

Для вызова хранимой функций и передачи в неё параметров так же был реализован API на ЯП Scala, но на этот раз при помощи SQL интерполятора, поскольку в Type-safe DSL данная возможность не реализована.

На вход подаются множества входных и выходных ЗУНов, а выходом является множество курсов, которые могут служить для построение траекторий по заданным входным и выходным множествам:

```
def makeTrajectory(inputKnowledge: Seq[KnowledgeEntity] = Seq.empty,
                   outputKnowledge: Seq[KnowledgeEntity] = Seq.empty,
                   inputAbilities: Seq[AbilityEntity] = Seq.empty,
                   outputAbilities: Seq[AbilityEntity] = Seq.empty,
                   inputSkills: Seq[SkillEntity] = Seq.empty,
                   outputSkill: Seq[SkillEntity] = Seq.empty,
                  ): Seq[CourseEntity] = {
```

Затем происходит вызов хранимой функции и передача в нее параметров. Поскольку на вход даются множества объектов, а алгоритм принимает массивы, то из каждого объекта переданных множеств берется id, а затем из полученных id формируется SQL массив:

```
sql"""
    SELECT * FROM makeTrajectory(
      inputsknowledge:= array[${inputKnowledge.map(_.id)}]::UUID[],
      outputsknowledge:= array[${outputKnowledge.map(_.id)}]::UUID[],
      inputsabilities := array[${inputAbilities.map(_.id)}]::UUID[],
      outputsabilities := array[${outputAbilities.map(_.id)}]::UUID[],
      inputsskills := array[${inputSkills.map(_.id)}]::UUID[],
      outputsskills := array[${outputSkill.map(_.id)}]::UUID[]);
 """
```

После завершения работы алгоритма, по полученным значениям собираются Scala объекты.

6.2. Тестирование алгоритма

Для тестирования алгоритма был подготовлен специальный набор данных, т.к. тестирование на случайно сгенерированных данных представляется невозможным из-за высокой сложности проверки существующих связей человеком:

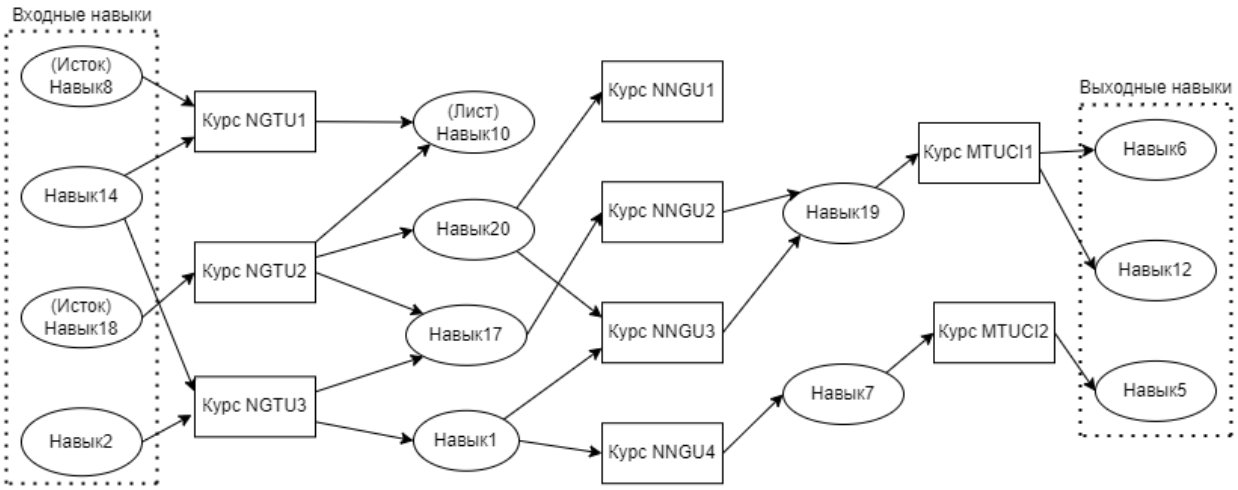

Рисунок 16. - Набор данных для тестирования

Для упрощения рассмотрим работу алгоритма только в плоскости Навыков (Skills).

Алгоритм поиска в ширину начинается от выходных навыков (Рис. 17):

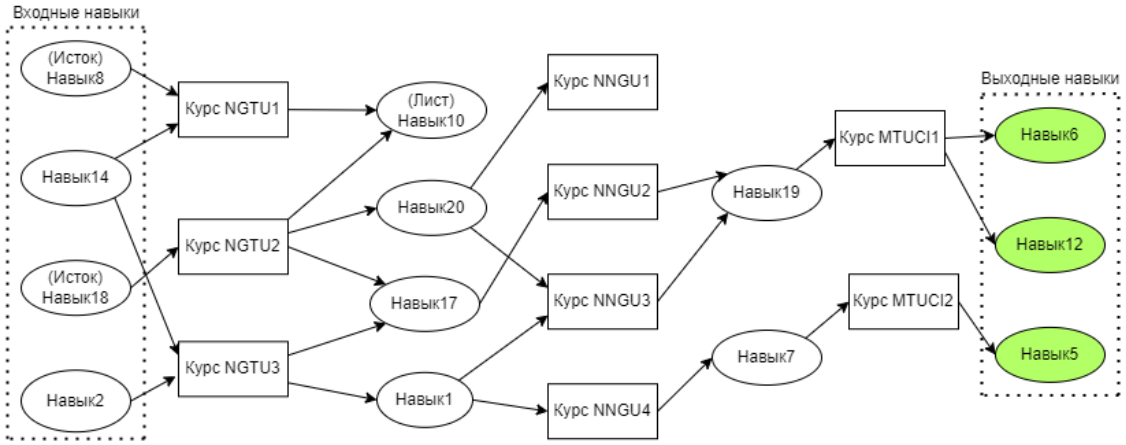

Рисунок 17. - Шаг 1

Для каждого из навыков находим курсы, для которых они являются выходными (Рис. 18):

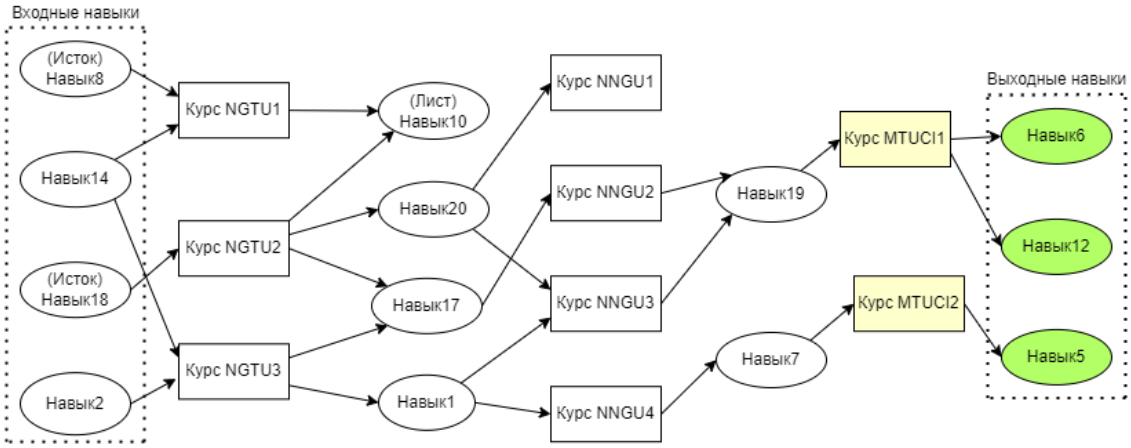

Рисунок 18. - Шаг 2

Для каждого из найденных курсов находим входные навыки (Рис. 19):

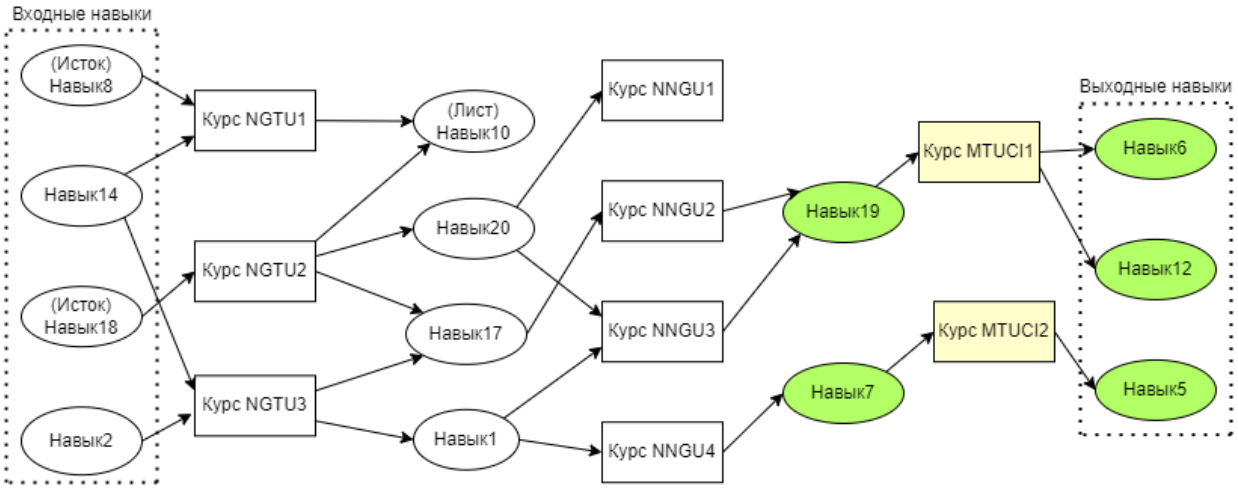

Рисунок 19. - Шаг 3

Для каждого из навыков находим курсы, для которых они являются выходными (Рис. 20):

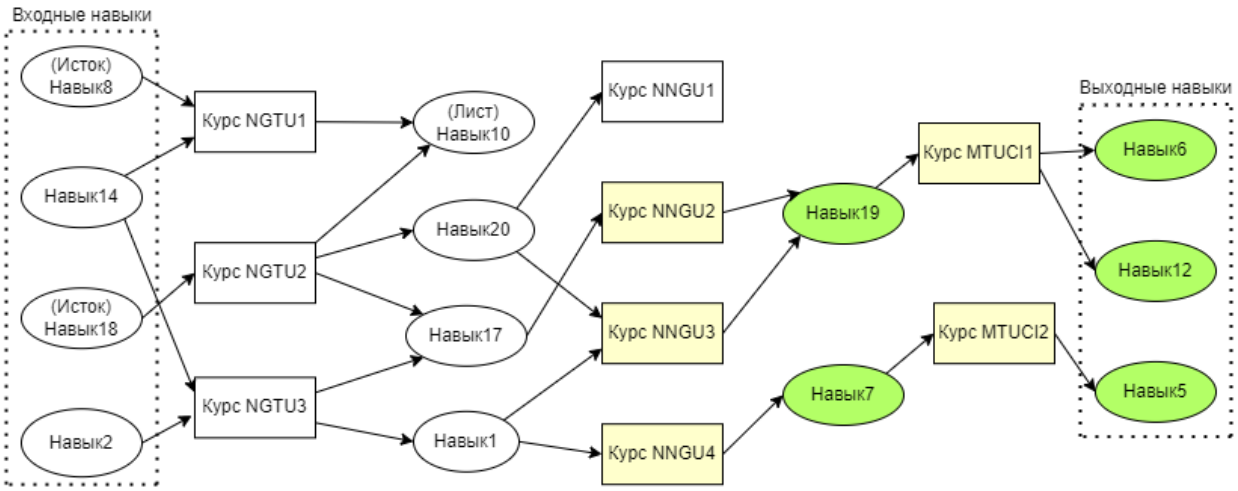

Рисунок 20. - Шаг 4

Повторяя данные шаги в итоге, получим следующую картину (Рис. 21):

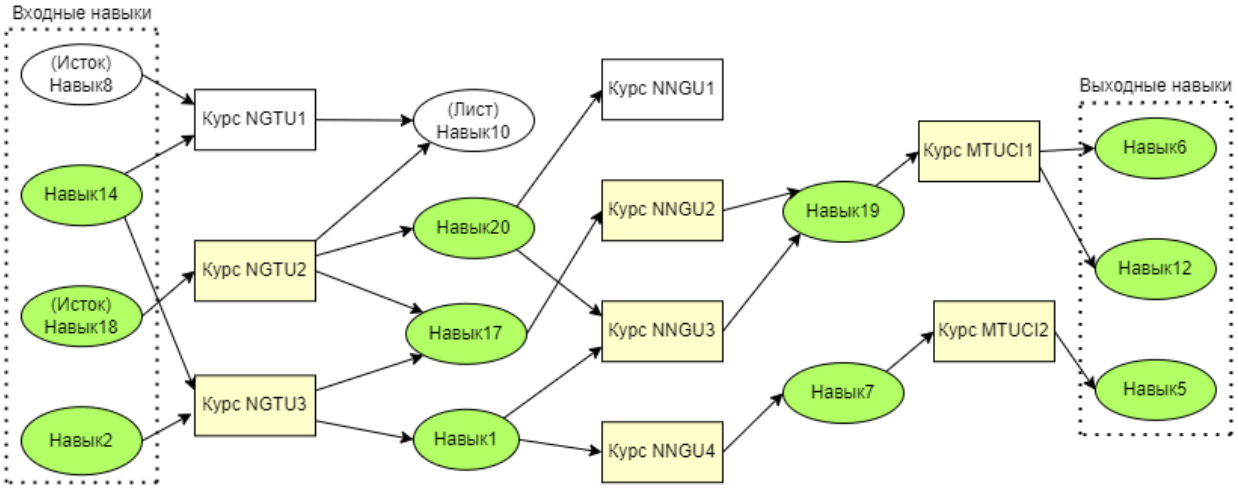

Рисунок 21. - Итог

Таким образом мы построили траектории от выходных навыков к входным, игнорируя те части, которые не позволяют построить траектории. Так же игнорируются излишние входные навыки, поданные на вход алгоритму.

6.3. Тестирование программной реализации

Результаты, полученные алгоритмом для схемы, приведенной выше:

```
Входные навыки: List(Навык8, Навык14, Навык18, Навык2)
Выходные навыки: List(Навык6, Навык12, Навык5)
Множество курсов для построение траектории: List(Курс MTUCI2,
Курс NGTU2, Курс MTUCI1, Курс NNGU2, Курс NNGU3, Курс NNGU4, Курс
NGTU3)
```

Если невозможно построить траекторию, то возвращается пустое множество:

```
Входные навыки: List(Навык8, Навык14, Навык22, Навык2)
выходные навыки: List(Навык30, Навык29, Навык28, Навык27)
Множество курсов для построение траектории: List()
```

Так же были проведены тесты по множествам знаний и умений и всех множеств сразу, которые не вошли пояснительную записку из-за сложности доказательства, что построенные траектории являются верными. На всех тестах алгоритмах показал верные результаты.

7. Выводы и заключение

В ходе данной работы была спроектирована и реализована распределённая база данных на основе СУБД PostgreSQL, разработан и реализован алгоритм построения сетевых траекторий обучения на процедурном языке PL/pgSQL. Для взаимодействия с БД и алгоритмом был реализован внешний API на ЯП Scala, при помощи библиотеки ScalikeJDBC.

Реализованная распределённая база данных является прототипом и имеет большой недостаток: работа с несколькими БД при помощи расширения postgres_fdw осложняет дальнейшее масштабирование и включение новых ВУЗов в систему, т.к. при каждом новом узле необходимо подключить его к общему интерфейсу, а также изменить алгоритм с учетом нового узла. Решением данной проблемы может служить переход на Postgres-XL, где все базы на разных узлах по умолчанию объединяются общим интерфейсом, а таблицы с одинаковыми имена и наборами столбцов на разных узлах считаются одной таблицей распределённой системы. Таким образом, в запросах нет необходимости указывать к какой именно БД и на каком узле идет обращение.

Следующим шагом развития данной работы является создание системы метрик, исследования эвристик, которые позволят повысить эффективность средств хранения сведений о курсах в распределенной БД.

В дальнейшем планируется развить внешний API доступа к БД до веб-сервиса и предоставить удобный пользовательский интерфейс. Так же при увеличении данных, хранимых в БД, будет проведено индексирование таблицы для увеличения производительности.

8. Список используемых источников

1. *Тарасов А. В.* Алгоритм генерации граф-модели образовательных траекторий на основе объектной декомпозиции компетенций выпускника / *А. В. Тарасов, В. О. Сапожников, Д. В. Жевнерчук, П. С. Кулясов* // Информационные технологии в науке, промышленности и образовании. Всероссийская научно-техническая конференция. - Ижевс: ИжГТУ имени М. Т. Калашникова, 2022.
2. PostgresPro: Документация по расширению процедурного языка SQL для СУБД Postgres – PL/pgSQL – URL: https://postgrespro.ru/docs/postgresql/9.6/plpgsql
3. *Мартин Одерски*. Scala. Профессиональное программирование 4-е издание / *Мартин Одерски, Лекс Спун, Билл Веннерс;* пер. на русский язык ООО Издательство «Питер», 2021г.
4. ScalikeJDBC: Documentation – URL: http://scalikejdbc.org/

Приложение А.
Скрипт создания общего интерфейса для распределенной базы данных

```
CREATE EXTENSION postgres_fdw;
-- Создаём схемы в которых будут хранится
-- представления удаленных таблицы
CREATE SCHEMA nngu_courses;
CREATE SCHEMA mtuci_courses;

------------------------------------------------------------
--              Подключение к серверу mtuci               --
------------------------------------------------------------
/*
    Создание стороннего сервера для покдключения к БД MTUCI
*/
CREATE SERVER mtuci_server
    FOREIGN DATA WRAPPER postgres_fdw
    OPTIONS (host 'localhost', port '5432', dbname 'mtuci');

/*
    Сопоставление ролей пользователя Postgres текущей БД и БД MTUCI
*/
CREATE USER MAPPING FOR postgres
        SERVER mtuci_server
        OPTIONS (user 'postgres', password 'postgre');

/*
    Импорт представления таблиц из схемы courses БД MTUCI
    в схему mtuci_courses текущей БД
*/
IMPORT FOREIGN SCHEMA courses
    FROM SERVER mtuci_server INTO mtuci_courses;


------------------------------------------------------------
--              Подключение к серверу nngu                --
------------------------------------------------------------
/*
    Создание стороннего сервера для покдключения к БД nngu
*/
CREATE SERVER nngu_server
    FOREIGN DATA WRAPPER postgres_fdw
    OPTIONS (host 'localhost', port '5432', dbname 'nngu');

/*
    Сопоставление ролей пользователя Postgres текущей БД и БД MTUCI
*/
CREATE USER MAPPING FOR postgres
        SERVER nngu_server
        OPTIONS (user 'postgres', password 'postgre');

/*
    Импорт представления таблиц из схемы courses БД MTUCI
    в схему mtuci_courses текущей БД
*/
IMPORT FOREIGN SCHEMA courses
    FROM SERVER nngu_server INTO nngu_courses;
```

Приложение Б.
Скрипт алгоритма построения сетевых траекторий обучения

```
/*
    Функция нахождения разности массивов.
    Из массива minuend вычитаются все значения из массива subtrahend
*/
CREATE OR REPLACE FUNCTION arrDif(minuend anyarray, subtrahend anyarray)
RETURNS anyarray AS
$code$
BEGIN
    RETURN (SELECT array
                        (SELECT unnest(minuend)
                         EXCEPT
                         SELECT unnest(subtrahend)));
END;
$code$ LANGUAGE plpgsql;



/*
    Получение курсов по id из переданного массива
*/
CREATE OR REPLACE FUNCTION getCoursesFromArray(idArr UUID[])
    RETURNS TABLE
            (
                id   UUID,
                name VARCHAR(255)
            )
AS
$code$
BEGIN
    RETURN QUERY
        (SELECT courses.course.id, courses.course.name
         FROM courses.course
         WHERE courses.course.id IN (SELECT * FROM
                        unnest(idArr))
         UNION
         SELECT nngu_courses.course.id,
                        nngu_courses.course.name
         FROM  nngu_courses.course
         WHERE  nngu_courses.course.id IN (SELECT * FROM
                        unnest(idArr))
         UNION
         SELECT mtuci_courses.course.id,
                        mtuci_courses.course.name
         FROM mtuci_courses.course
         WHERE mtuci_courses.course.id IN (SELECT * FROM unnest(idArr)));
END;
$code$ LANGUAGE plpgsql;
```

```
-------------------------------------------------------------------------------
--                Функции получения курсов по выходным ЗУНам                 --
-------------------------------------------------------------------------------

/*
Получение массива id курсов, для которых переданный Навык (Skill) является
выходным
*/
CREATE OR REPLACE FUNCTION getCoursesByOutputSKills(kasID UUID) RETURNS
UUID[] AS
$code$
DECLARE
courses UUID[];

/*
В результирующий массив добавляем id курсов из разных БД
*/
BEGIN
courses := array_cat(courses, ARRAY(SELECT courses.course.id
                                    FROM courses.course
                                             LEFT JOIN
courses.course_output_skill
                                                       ON courses.course.id =
courses.course_output_skill.course_id
                                    WHERE courses.course_output_skill.skill_id =
kasID));

courses := array_cat(courses, ARRAY(SELECT nngu_courses.course.id
                                    FROM nngu_courses.course
                                             LEFT JOIN
nngu_courses.course_output_skill
                                                       ON nngu_courses.course.id
= nngu_courses.course_output_skill.course_id
                                    WHERE
nngu_courses.course_output_skill.skill_id = kasID));

courses := array_cat(courses, ARRAY(SELECT mtuci_courses.course.id
                                    FROM mtuci_courses.course
                                             LEFT JOIN
mtuci_courses.course_output_skill
                                                       ON mtuci_courses.course.id
= mtuci_courses.course_output_skill.course_id
                                    WHERE
mtuci_courses.course_output_skill.skill_id = kasID));

RETURN courses;
END;
$code$ LANGUAGE plpgsql;

/*
Получение массива id курсов, для которых переданное Умение (Ability) является
выходным
*/
CREATE OR REPLACE FUNCTION getCoursesByOutputAbilities(kasID UUID) RETURNS
UUID[] AS
$code$
DECLARE
courses UUID[];

/*
В результирующий массив добавляем id курсов из разных БД
*/
```

```
BEGIN
courses := array_cat(courses, ARRAY(SELECT courses.course.id
                                    FROM courses.course
                                             LEFT JOIN
courses.course_output_ability
                                                       ON courses.course.id =
courses.course_output_ability.course_id
                                    WHERE
courses.course_output_ability.ability_id = kasID));

courses := array_cat(courses, ARRAY(SELECT nngu_courses.course.id
                                    FROM nngu_courses.course
                                             LEFT JOIN
nngu_courses.course_output_ability
                                                       ON nngu_courses.course.id
= nngu_courses.course_output_ability.course_id
                                    WHERE
nngu_courses.course_output_ability.ability_id = kasID));

courses := array_cat(courses, ARRAY(SELECT mtuci_courses.course.id
                                    FROM mtuci_courses.course
                                             LEFT JOIN
mtuci_courses.course_output_ability
                                                       ON mtuci_courses.course.id
= mtuci_courses.course_output_ability.course_id
                                    WHERE
mtuci_courses.course_output_ability.ability_id = kasID));

RETURN courses;
END;
$code$ LANGUAGE plpgsql;

/*
Получение массива id курсов, для которых переданное Знание (Knowledge)
является выходным
*/
CREATE OR REPLACE FUNCTION getCoursesByOutputKnowledge(kasID UUID) RETURNS
UUID[] AS
$code$
DECLARE
courses UUID[];

/*
В результирующий массив добавляем id курсов из разных БД
*/
BEGIN
courses := array_cat(courses, ARRAY(SELECT courses.course.id
                                    FROM courses.course
                                             LEFT JOIN
courses.course_output_knowledge
                                                       ON courses.course.id =
courses.course_output_knowledge.course_id
                                    WHERE
courses.course_output_knowledge.knowledge_id = kasID));

courses := array_cat(courses, ARRAY(SELECT nngu_courses.course.id
                                    FROM nngu_courses.course
                                             LEFT JOIN
nngu_courses.course_output_knowledge
                                                       ON nngu_courses.course.id
= nngu_courses.course_output_knowledge.course_id
                                    WHERE
nngu_courses.course_output_knowledge.knowledge_id = kasID));
```

```
courses := array_cat(courses, ARRAY(SELECT mtuci_courses.course.id
                                    FROM mtuci_courses.course
                                             LEFT JOIN
mtuci_courses.course_output_knowledge
                                                       ON mtuci_courses.course.id
= mtuci_courses.course_output_knowledge.course_id
                                    WHERE
mtuci_courses.course_output_knowledge.knowledge_id = kasID));

RETURN courses;
END;
$code$ LANGUAGE plpgsql;



-------------------------------------------------------------------------------
--           Функции получения входных ЗУНов переданного Курса               --
-------------------------------------------------------------------------------

/*
Получение массива id Навыков (Skills), которые являются входным для
переданного курса
*/
CREATE OR REPLACE FUNCTION getInputSkillsByCourse(courseID UUID) RETURNS
UUID[] AS
$code$
DECLARE
skills UUID[];

/*
В результирующий массив добавляем id Навыков(Skill) из разных БД
*/
BEGIN
skills := array_cat(skills, ARRAY(SELECT courses.skill.id
                                  FROM courses.skill
                                           LEFT JOIN courses.course_input_skill
                                                     ON courses.skill.id =
courses.course_input_skill.skill_id
                                  WHERE courses.course_input_skill.course_id =
courseID));

skills := array_cat(skills, ARRAY(SELECT nngu_courses.skill.id
                                  FROM nngu_courses.skill
                                           LEFT JOIN
nngu_courses.course_input_skill
                                                     ON nngu_courses.skill.id =
nngu_courses.course_input_skill.skill_id
                                  WHERE nngu_courses.course_input_skill.course_id
= courseID));

skills := array_cat(skills, ARRAY(SELECT mtuci_courses.skill.id
                                  FROM mtuci_courses.skill
                                           LEFT JOIN
mtuci_courses.course_input_skill
                                                     ON mtuci_courses.skill.id =
mtuci_courses.course_input_skill.skill_id
                                  WHERE
mtuci_courses.course_input_skill.course_id = courseID));

RETURN skills;
END;
$code$ LANGUAGE plpgsql;
```

```
/*
Получение массива id Умений (Abilities), которые являются входным для
переданного курса
*/
CREATE OR REPLACE FUNCTION getInputAbilitiesByCourse(courseID UUID) RETURNS
UUID[] AS
$code$
DECLARE
abilities UUID[];

/*
В результирующий массив добавляем id Умений (Abilities) из разных БД
*/
BEGIN
abilities := array_cat(abilities, ARRAY(SELECT courses.ability.id,
courses.ability.name
                                        FROM courses.ability
                                                 LEFT JOIN
courses.course_input_ability
                                                           ON courses.ability.id
= courses.course_input_ability.ability_id
                                        WHERE
courses.course_input_ability.course_id = courseID));

abilities := array_cat(abilities, ARRAY(SELECT nngu_courses.ability.id,
nngu_courses.ability.name
                                        FROM nngu_courses.ability
                                                 LEFT JOIN
nngu_courses.course_input_ability
                                                           ON
nngu_courses.ability.id = nngu_courses.course_input_ability.ability_id
                                        WHERE
nngu_courses.course_input_ability.course_id = courseID));

abilities := array_cat(abilities, ARRAY(SELECT mtuci_courses.ability.id,
mtuci_courses.ability.name
                                        FROM mtuci_courses.ability
                                                 LEFT JOIN
mtuci_courses.course_input_ability
                                                           ON
mtuci_courses.ability.id = mtuci_courses.course_input_ability.ability_id
                                        WHERE
mtuci_courses.course_input_ability.course_id = courseID));

RETURN abilities;
END;
$code$ LANGUAGE plpgsql;

/*
Получение массива id Знаний (Knowledge), которые являются входным для
переданного курса
*/
CREATE OR REPLACE FUNCTION getInputKnowledgeByCourse(courseID UUID) RETURNS
UUID[] AS
$code$
DECLARE
knowledge UUID[];

/*
В результирующий массив добавляем id Умений (Abilities) из разных БД
*/
BEGIN
```

```
knowledge := array_cat(knowledge, (SELECT courses.knowledge.id,
courses.knowledge.name
                                FROM courses.knowledge
                                        LEFT JOIN
courses.course_input_knowledge
                                                ON courses.knowledge.id =
courses.course_input_knowledge.knowledge_id
                                WHERE courses.course_input_knowledge.course_id
= courseID));

knowledge := array_cat(knowledge, (SELECT nngu_courses.knowledge.id,
nngu_courses.knowledge.name
                                FROM nngu_courses.knowledge
                                        LEFT JOIN
nngu_courses.course_input_knowledge
                                                ON
nngu_courses.knowledge.id = nngu_courses.course_input_knowledge.knowledge_id
                                WHERE
nngu_courses.course_input_knowledge.course_id = courseID));

knowledge := array_cat(knowledge, (SELECT mtuci_courses.knowledge.id,
mtuci_courses.knowledge.name
                                FROM mtuci_courses.knowledge
                                        LEFT JOIN
mtuci_courses.course_input_knowledge
                                                ON
mtuci_courses.knowledge.id =
mtuci_courses.course_input_knowledge.knowledge_id
                                WHERE
mtuci_courses.course_input_knowledge.course_id = courseID));

RETURN knowledge;
END;
$code$ LANGUAGE plpgsql;


-------------------------------------------------------------------------
--       Функции построение траекторий для отдельных видов KASов           --
-------------------------------------------------------------------------

/*
Построение траектории по Навыкам (Skills)
От выходных Навыков (outputs) стороются пути к входным (inputs)

Возращается массив id курсов, через которые строится траектория
*/
CREATE OR REPLACE FUNCTION makeTrajectoryBySkills(inputs UUID[], outputs
UUID[]) RETURNS UUID[] AS
$code$
DECLARE
courses    UUID[];
tempArr    UUID[];
tempCourse UUID;
tempKas    UUID;

BEGIN
-- Поскольку plpgsql не поддерживает работу с многомерными массивами,
-- используем временную таблицу, в которую будем помещать массивы.
CREATE TEMP TABLE steps
(
stepNumber SERIAL,
step       UUID[]
);
```

```
-- Первый шаг алгоритма выполняется от переданных выходных Навыков (outputs)
INSERT INTO steps(step)
VALUES (outputs);

-- Цикл - Пока есть следующий шаг
WHILE (NOT ((SELECT step FROM steps ORDER BY stepNumber DESC LIMIT 1) =
'{}'))
LOOP
-- Объявление локальный переменных внутри цикла.
-- Обнуляются при каждой иттерации
    DECLARE
        coursesByStep UUID[];
        nextStep      UUID[];

    BEGIN
        -- Для каждого Навыка (tempKas) из последнего шага
        -- получаем Курсы, для которых данный Навык является выходным
        -- и записываем в массив курсов данного шага
        FOREACH tempKas IN ARRAY (SELECT step FROM steps ORDER BY stepNumber
DESC LIMIT 1)
            LOOP
                tempArr := (SELECT * FROM getCoursesByOutputSKills(tempKas));
                coursesByStep := array_cat(coursesByStep, tempArr);
            END LOOP;

        -- Если мы не смогли найти курсов, для которых Навык (skill) является
выходным
        -- значит, что данные Навыки (Skills) являются истоками
        -- Работа алгоритма прерывается, возвращается пустая таблица
        IF (coursesByStep = '{}') THEN
            DROP TABLE steps;
            RAISE EXCEPTION 'Невозможно построить траекторию по Навыкам
(Skills)';
        END IF;

        -- Для каждого курса, полученного на данном шаге получаем входные
Навыки (Skills)
        -- и записываем их в следующий шаг
        FOREACH tempCourse IN ARRAY (coursesByStep)
            LOOP
                nextStep := array_cat(nextStep, (SELECT * FROM
getInputSkillsByCourse(tempCourse)));
            END LOOP;

        -- В следующий шаг записываем разность полученных выше Навыков
(Skills)
        -- и входных Навыков (inputs)
        -- Таким образом мы останавливаем работу алгоритма на уже построенных
траекториях
        INSERT INTO steps(step)
        VALUES ((SELECT * FROM arrDif(nextStep, inputs)));

        -- В результирующий массив курсов записываем курсы, полученные на
данном шаге
        courses := array_cat(courses, coursesByStep);
    END;
END LOOP;

DROP TABLE steps;

-- Возвращаем массив id курсов
RETURN courses;
```

```
END;
$code$ LANGUAGE plpgsql;

/*
Построение траектории по Умениям (Abilities)
От выходных Умений (outputs) стороются пути к входным (inputs)

Возращается таблица курсов, через которые строится траектория
*/
CREATE OR REPLACE FUNCTION makeTrajectoryByAbilities(inputs UUID[], outputs
UUID[]) RETURNS UUID[] AS
$code$
DECLARE
courses    UUID[];
tempArr    UUID[];
tempCourse UUID;
tempKas    UUID;

BEGIN
-- Поскольку plpgsql не поддерживает работу с многомерными массивами,
-- используем временную таблицу, в которую будем помещать массивы.
CREATE TEMP TABLE steps
(
stepNumber SERIAL,
step       UUID[]
);

-- Первый шаг алгоритма выполняется от переданных выходных Умений (Abilities)
INSERT INTO steps(step)
VALUES (outputs);

-- Цикл - Пока есть следующий шаг
WHILE (NOT ((SELECT step FROM steps ORDER BY stepNumber DESC LIMIT 1) =
'{}'))
LOOP
-- Объявление локальный переменных внутри цикла.
-- Обнуляются при каждой иттерации
    DECLARE
        coursesByStep UUID[];
        nextStep      UUID[];

    BEGIN
        -- Для каждого Умения (tempKas) из последнего шага
        -- получаем Курсы, для которых данное Умение является выходным
        -- и записываем в массив курсов данного шага
        FOREACH tempKas IN ARRAY (SELECT step FROM steps ORDER BY stepNumber
DESC LIMIT 1)
            LOOP
                tempArr := (SELECT * FROM
getCoursesByOutputAbilities(tempKas));
                coursesByStep := array_cat(coursesByStep, tempArr);
            END LOOP;

        -- Если мы не смогли найти курсов, для которых Умение (ability)
является выходным
        -- значит, что данные Умения (Abilities) являются истоками
        -- Работа алгоритма прерывается, возвращается пустая таблица
        IF (coursesByStep = '{}') THEN
            DROP TABLE steps;
            RAISE EXCEPTION 'Невозможно построить траекторию по Умениям
(Abilities)';
        END IF;
```

```
        -- Для каждого курса, полученного на данном шаге получаем входные
Умения (Abilities)
        -- и записываем их в следующий шаг
        FOREACH tempCourse IN ARRAY (coursesByStep)
            LOOP
                nextStep := array_cat(nextStep, (SELECT * FROM
getInputAbilitiesByCourse(tempCourse)));
            END LOOP;

        -- В следующий шаг записываем разность полученных выше Умений
(Abilities)
        -- и входных Умений (inputs)
        -- Таким образом мы останавливаем работу алгоритма на уже построенных
траекториях
        INSERT INTO steps(step)
        VALUES ((SELECT * FROM arrDif(nextStep, inputs)));

        -- В результирующий массив курсов записываем курсы, полученные на
данном шаге
        courses := array_cat(courses, coursesByStep);
    END;
END LOOP;

DROP TABLE steps;

-- Возвращаем массив id курсов
RETURN courses;
END;
$code$ LANGUAGE plpgsql;

/*
Построение траектории по Знаниям (Knowledge)
От выходных Знаний (outputs) стороются пути к входным (inputs)

Возращается таблица курсов, через которые строится траектория
*/
CREATE OR REPLACE FUNCTION makeTrajectoryByKnowledge(inputs UUID[], outputs
UUID[]) RETURNS UUID[] AS
$code$
DECLARE
courses    UUID[];
tempArr    UUID[];
tempCourse UUID;
tempKas    UUID;

BEGIN
-- Поскольку plpgsql не поддерживает работу с многомерными массивами,
-- используем временную таблицу, в которую будем помещать массивы.
CREATE TEMP TABLE steps
(
stepNumber SERIAL,
step       UUID[]
);

-- Первый шаг алгоритма выполняется от переданных выходных Знаний (outputs)
INSERT INTO steps(step)
VALUES (outputs);

-- Цикл - Пока есть следующий шаг
WHILE (NOT ((SELECT step FROM steps ORDER BY stepNumber DESC LIMIT 1) =
'{}'))
LOOP
-- Объявление локальный переменных внутри цикла.
```

```
-- Обнуляются при каждой иттерации
    DECLARE
        coursesByStep UUID[];
        nextStep      UUID[];

    BEGIN
        -- Для каждого Знания (tempKas) из последнего шага
        -- получаем Курсы, для которых данное Знание является выходным
        -- и записываем в массив курсов данного шага
        FOREACH tempKas IN ARRAY (SELECT step FROM steps ORDER BY stepNumber
DESC LIMIT 1)
            LOOP
                tempArr := (SELECT * FROM
getCoursesByOutputKnowledge(tempKas));
                coursesByStep := array_cat(coursesByStep, tempArr);
            END LOOP;

        -- Если мы не смогли найти курсов, для которых Знание (knowledge)
является выходным
        -- значит, что данные Знания (Knowledge) являются истоками
        -- Работа алгоритма прерывается, возвращается пустая таблица
        IF (coursesByStep = '{}') THEN
            DROP TABLE steps;
            RAISE EXCEPTION 'Невозможно построить траекторию по Знаниям
(Knowledge)';
        END IF;

        -- Для каждого курса, полученного на данном шаге получаем входные
Знания (Knowledge)
        -- и записываем их в следующий шаг
        FOREACH tempCourse IN ARRAY (coursesByStep)
            LOOP
                nextStep := array_cat(nextStep, (SELECT * FROM
getInputKnowledgeByCourse(tempCourse)));
            END LOOP;

        -- В следующий шаг записываем разность полученных выше Знаний
(Knowledge)
        -- и входных Знаний (inputs)
        -- Таким образом мы останавливаем работу алгоритма на уже построенных
траекториях
        INSERT INTO steps(step)
        VALUES ((SELECT * FROM arrDif(nextStep, inputs)));

        -- В результирующий массив курсов записываем курсы, полученные на
данном шаге
        courses := array_cat(courses, coursesByStep);
    END;
END LOOP;

DROP TABLE steps;

-- Возвращаем массив id курсов
RETURN courses;
END;
$code$ LANGUAGE plpgsql;

/*
Получение курсов для постоения траектории по заданным входным и выходным
КАSам
*/
CREATE OR REPLACE FUNCTION makeTrajectory(inputsKnowledge UUID[],
outputsKnowledge UUID[],
```

```
                                    inputsAbilities UUID[], outputsAbilities
UUID[],
                                    inputsSkills UUID[], outputsSkills UUID[])
RETURNS TABLE
    (
        id   UUID,
        name VARCHAR(255)
    )
AS
$code$
DECLARE
resultCourses UUID[];

BEGIN
IF NOT (inputsKnowledge = '{}' AND outputsKnowledge = '{}') THEN
resultCourses := array_cat(resultCourses,
                           (SELECT * FROM
makeTrajectoryByKnowledge(inputsKnowledge, outputsKnowledge)));
END IF;

IF NOT (inputsAbilities = '{}' AND outputsAbilities = '{}') THEN
resultCourses := array_cat(resultCourses,
                           (SELECT * FROM
makeTrajectoryByAbilities(inputsAbilities, outputsAbilities)));
END IF;

IF NOT (inputsSkills = '{}' AND outputsSkills = '{}') THEN
resultCourses := array_cat(resultCourses,
                           (SELECT * FROM
makeTrajectoryBySkills(inputsSkills, outputsSkills)));
END IF;

RETURN QUERY
    (SELECT DISTINCT * FROM getCoursesFromArray(resultCourses));

EXCEPTION
WHEN OTHERS
THEN
    RETURN;
END;
$code$ LANGUAGE plpgsql;
```