Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К К ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЕ

НА ТЕМУ:

«Метод построения поисковых индексов в реляционной базе данных на основе глубоких нейронных сетей»

Студент: ИУ7-83Б _____ М. Д. Маслова
(группа) (подпись, дата) (И. О. Фамилия)

Руководитель: _____ А. А. Оленев
(подпись, дата) (И. О. Фамилия)

Нормоконтролер: _____ Д. Ю. Мальцева
(подпись, дата) (И. О. Фамилия)

2023 г.

1 Технологическая часть

1.1 Выбор средств программной реализации

Для реализации метода построения индекса в реляционной базе данных на основе глубоких нейроных сетей в качестве языка программирования выбран Python 3.10 [**python**], так как предоставляет широкий выбор библиотек для глубокого обучения и визуализации его результатов. В качестве библиотеки глубокого обучения выбран TensorFlow 2.11.0 [**tf**] и работающий поверх нее высокоуровневый программный интерфейс Keras 2.11.0 [**keras**]. Для работы с массивами данных выбрана библиотека numpy [**numpy**].

В качестве реляционной системы управления базами данных выбрана SQLite [**sqlite**], предоставляющая программный интерфейс виртуальных таблиц, позволяющих релизовать пользовательский поисковый индекс. Виртуальные таблицы являются одним из видов расширений SQLite, программный интерфейс которых предоставляется на языке C [**c**], который и выбран в качестве языка программирования для взаимодействия с реляционной базой данных.

Для обеспечения взаимодействия между компонентом работы с базой данных и компонентом, непосредственно реализующий индекс, используются библиотеки языка C `Python.h` для работы с объектами языка Python и `numpy/arrayobject.h`, предоставляющая программный интерфейс для работы с numpy-массивами, которые являются основным типом данных, через который происходит взаимодействие модулей.

1.2 Реализация программного обеспечения

1.2.1 Форматы входных и выходных данных

Основой для построения индекса в качестве входных выступают данные из таблицы реляционной базы данных SQLite. Требованием к таблице является наличие атрибута целочисленного типа (`INTEGER`) с уникальными значениями (`UNIQUE`).

Для создания индекса в аргументах соответствующего запроса должен присутствовать идентификатор столбца, удовлетворяющего требованию описанному выше, и строковое имя модели индекса, на основе которой он строится (`FCNN2` — для модели с двумя скрытыми слоями, `FCNN3` — с тремя).

В качестве входных данных компонента, реализующего индекса, является набор значений столбца, идентификатор которого был указан в аргументах запроса на создание индекса, и идентификаторы строк `ROWID` индексируемой таблицы.

Выходным данных является указатель на объект, представляющий индекс на основе глубокой нейронной сети, обученный на предоставленных входных данных.

Формат входных и выходных данных операций с индексом (поиска и вставки) описан в следующем пункте.

1.2.2 Поддерживаемые виды запросов

Разработанное программное обеспечения предоставляет возможность работы только с запросами фильтрации, представленными оператором `WHERE` следующими со следующими условиями:

— `column operator value`,
  где `column` — имя проиндексированного столбца,
  `operator` — одна из операций сравнения: `=, <, >, <=, >=`,
  `value` — некоторое целочисленное значение.
— `column BETWEEN value1 AND value2`,
  где `column` — имя проиндексированного столбца,
  `value1, value2` — целочисленные значения, представляющие нижнюю и верхнюю границы диапазона.

Выходным значением из модуля на языке Python, реализующего индекс, по данным запросам является numpy-массив с соответствующими запросу значениями ROWID, а результатом работы программного обеспечения — набор записей таблицы с ROWID из представленного массива.

Для вставки поддерживается стандартный запрос `INSERT`, результатом которого является индекс с переобученной на новых данных моделью. Запросы удаления и изменения не поддерживаются, так как являются вторичными для оценки работы метода.

1.2.3 Программный интерфейс виртуальных таблиц

Виртуальные таблицы SQLite [**vtable**] — это объект базы данных, представляющий с точки зрения инструкций SQL обычную таблицу или представле-

ние, но обрабатывающий запросы посредством вызова функций программного интерфейса, которые реализуются пользователем.

Виртальные таблицы являются расширением SQLite, регистрация которых происходит с использованием макросов и фукнции инициализации, представленных на линстинге 1.1.

Листинг 1.1 — Инициализация расширения

```
#include <sqlite3ext.h>

SQLITE_EXTENSION_INIT1

int sqlite3_extension_init(sqlite3 *db,
                           char **pzErrMsg,
                           const sqlite3_api_routines *pApi)
{
  int rc = SQLITE_OK;
  SQLITE_EXTENSION_INIT2(pApi);

  /* инициализация расширения */

  return rc;
}
```

При инициализация расширения вирутальной таблицы должен быть зарегистрирован модуль, описывающийся структурой `sqlite3_module`, с помощью функции регистрации `sqlite3_create_module`, подробное описание которых представлено на листинге 1.2

Листинг 1.2 — Структура и функции для регистрации модуля виртуальной таблицы

```
struct sqlite3_module {
    int iVersion;
    int (*xCreate)(sqlite3*, void *pAux,
                int argc, char *const*argv,
                sqlite3_vtab **ppVTab,
                char **pzErr);
    int (*xConnect)(sqlite3*, void *pAux,
                int argc, char *const*argv,
                sqlite3_vtab **ppVTab,
                char **pzErr);
    int (*xBestIndex)(sqlite3_vtab *pVTab, sqlite3_index_info*);
    int (*xDisconnect)(sqlite3_vtab *pVTab);
```

Продолжение листинга 1.2

```
13      int (*xDestroy)(sqlite3_vtab *pVTab);
14      int (*xOpen)(sqlite3_vtab *pVTab,
15                   sqlite3_vtab_cursor **ppCursor);
16      int (*xClose)(sqlite3_vtab_cursor*);
17      int (*xFilter)(sqlite3_vtab_cursor*,
18                     int idxNum, const char *idxStr,
19                     int argc, sqlite3_value **argv);
20      int (*xNext)(sqlite3_vtab_cursor*);
21      int (*xEof)(sqlite3_vtab_cursor*);
22      int (*xColumn)(sqlite3_vtab_cursor*, sqlite3_context*, int);
23      int (*xRowid)(sqlite3_vtab_cursor*, sqlite_int64 *pRowid);
24      int (*xUpdate)(sqlite3_vtab *, int,
25                     sqlite3_value **, sqlite_int64 *);
26      /* представлены указатели на реализующиеся методы */
27      /* при инициализации структуры, */
28      /* остальные поля принимают значение 0 */
29  };
30
31  int sqlite3_create_module(
32      sqlite3 *db,           /* соединение для регистрации модуля */
33      const char *zName,       /* имя модуля */
34      const sqlite3_module *, /* ссылка на структуру модуля */
35      void *,                  /* данные для xCreate/xConnect*/
36  );
```

Методы, сигнатуры которых представлены на листинге 1.2, можно разделить на две группы.

Методы для взаимодействие с таблицей, как с некоторым объектом, к которым относятся:

— `xCreate` — создание виртуальной таблицы, при выполнении соответствующего запроса, представленного на листинге 1.3;

— `xConnect` — подключение к виртуальной таблице, вызывающийся при выполнении любого запроса к таблице, который является первым при повторном подключении к базе данных;

— `xDestroy` — удаление вирутальной таблицы при выполнеивыполнении запроса, представленного на листинге 1.4;

— `xDisconnect` — удаление поключения к виртуальной таблице.

Листинг 1.3 — Запрос на создание виртуальной таблицы

```
1  CREATE VIRTUAL TABLE <имя_таблицы> USING <имя_модуля>(arg1, ...);
```

Листинг 1.4 — Запрос на удаление виртуальной таблицы

```
1  DROP TABLE <имя_таблицы>;
```

Данные методы работают со структурой `sqlite3_vtab`, представленной на листинге **??**. Для реализации нужных функциональностей указатель на данную структуру включается в пользовательскую, с которой уже работают представленные методы посредством преобразования типов. Это дает возможность передавать между методами вирутальной таблицы нужные данные.

Методы прохода по записям таблицы, использующие для этого структуру курсора `sqlite3_vtab_cursor`, представленую на листинге **??**, над которой также реализуют обертку для хранения необходимых для обработки переменных.

Данная группа представлена методом обработки вставки, удаления и изменения записи (последний) и методами для прохода по записям таблицы при поиске:

- `xOpen` — создание и инициализации структуры курсора;
- `xBestIndex` — получение параметров фильтрации и выбор лучшего индекса для обработки запроса;
- `xFilter` — получение соответствующих параметрам фильтрации записей, установка курсора на первую из них;
- `xEof` — проверка окончания списка выбранных записей;
- `xNext` — переход к следующей записи;
- `xColumn` — обработка столбца записи;
- `xClose` — удаление структуры курсора;
- `xUpdate` — реализация запросов вставки, удаления и изменения.

Для реализации метода построения индекса используются оберточные структуры для виртуальной таблицы и курсора, представленные на листинге **??**.

На листинге **??** представлена реализация метода создания индекса, реализованного в качестве `xCreate`. Код инициализации и запуска обучение индекса в Python через программиный интерфейс приведен на листинге **??**.

Обработка параметров фильтрации приведена на листинге **??**. Получение массива подходящих строк и проход для их вывода приведены на листингах **??**, **??** соответственно.

1.2.4 Реализация индекса

...

1.3 Сборка программного обеспечения

При сборке расширения SQLite компиляция и линковка происходит с флагами, обычно используемыми при сборке динамический библиотек: флаг `-fPIC` при компиляции в объектные файлы для создание позиционно-независимого кода и флаг `-shared` для получения файла динамической библиотеки. Дополнительными флагами при линковке являются флаги подключения библиотек `-lsqlite3` и `-lpython3.10`. Также при компиляции требуется указание путей к заголовочным файлам `Python.h` и `numpy/arrayobject.h`. Их автоматическое получения, а также ключевые моменты сборки приведены на листинге **??**.

Для работы компонента индекса, реализованного на Python, требуется установка зависимостей из уже сформированного файла `requirements.txt` путем выполения команды, представленной на листинге **??**. Также для штатной работы программного обеспечения требуется прописать путь к модулям, реализованным на языке Python, что приведено на том же листинге.

1.4 Взаимодействие с программным обеспечением

Взаимодействие с программным обеспечением проходит через командную строку sqlite3. Пример работы представлен на листинге **??**.

1.5 Результаты тестирования

После разработки программного обеспечение было проведно автоматическое тестирование модуля индекса на основе глубоких нейронных сетей. Были использованы следующие классы эквивалентности:

— поиск существующего единичного ключа;
— поиск наименьшего и наибольшего в наборе ключей;
— поиск несуществующего ключа;

— поиск ключей по условию < с существующим ключом в качестве границы;
— поиск ключей по условию < с несуществующим ключом в качестве границы;
— два предудыщих класса по условиям >, <=, >=;
— поиск по диапазону с двумя границами;
— поиск по диапазону с двумя границами, в который попадают все ключи;
— поиск по диапазону с двумя границами, в который не попадает ни один ключ.

Результаты тестирования представлены на листинге 1.5

Листинг 1.5 — Результаты автоматического тестирования

```
======== short test summary info ================
PASSED tests.py::TestLindex::test_train
PASSED tests.py::TestLindex::test_middle
PASSED tests.py::TestLindex::test_first
PASSED tests.py::TestLindex::test_last
PASSED tests.py::TestLindex::test_not_exist
PASSED tests.py::TestLindex::test_range_lw_in
PASSED tests.py::TestLindex::test_range_lw_out
PASSED tests.py::TestLindex::test_range_gr_in
PASSED tests.py::TestLindex::test_range_gr_out
PASSED tests.py::TestLindex::test_range_le_in
PASSED tests.py::TestLindex::test_range_le_out
PASSED tests.py::TestLindex::test_range_ge_in
PASSED tests.py::TestLindex::test_range_ge_out
PASSED tests.py::TestLindex::test_range_gle
PASSED tests.py::TestLindex::test_range_gle_all
PASSED tests.py::TestLindex::test_range_gle_none
PASSED tests.py::TestLindex::test_insert
======= 17 passed, 1 warning in 4.36s ===========
```

Также было проведено ручное интеграционное тестирвание программного обеспечения.