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.

СОДЕРЖАНИЕ

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 [1]. В качестве библиотеки глубокого обучения выбран Pytorch 2.0.0 [2]. Для работы с массивами данных выбрана библиотека numpy [3].

В качестве реляционной системы управления базами данных выбрана SQLite [4], 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 [5], 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 [6] — это объект базы данных, представляющий с точки зрения инструкций SQL обычную таблицу или представление, 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 функция 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);
    int (*xDestroy)(sqlite3_vtab *pVTab);
    int (*xOpen)(sqlite3_vtab *pVTab,
```

Продолжение листинга 1.2

```
                    sqlite3_vtab_cursor **ppCursor);
    int (*xClose)(sqlite3_vtab_cursor*);
    int (*xFilter)(sqlite3_vtab_cursor*,
                    int idxNum, const char *idxStr,
                    int argc, sqlite3_value **argv);
    int (*xNext)(sqlite3_vtab_cursor*);
    int (*xEof)(sqlite3_vtab_cursor*);
    int (*xColumn)(sqlite3_vtab_cursor*, sqlite3_context*, int);
    int (*xRowid)(sqlite3_vtab_cursor*, sqlite_int64 *pRowid);
    int (*xUpdate)(sqlite3_vtab *, int,
                    sqlite3_value **, sqlite_int64 *);
    /* представлены указатели на реализующиеся методы */
    /* при инициализации структуры, */
    /* остальные поля принимают значение 0 */
};

int sqlite3_create_module(
    sqlite3 *db,           /* соединение для регистрации модуля */
    const char *zName,       /* имя модуля */
    const sqlite3_module *, /* ссылка на структуру модуля */
    void *,                  /* данные для xCreate/xConnect*/
);
```

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

```
CREATE VIRTUAL TABLE <table_name> USING <module_name>(arg1, ...);
```

Листинг 1.4 — Запрос на удаление виртуальной таблицы

```
DROP TABLE <table_name>;
```

Данные методы работают со структурой `sqlite3_vtab`, представленной на листинге 1.5. 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.

Листинг 1.5 — Структура вирутальной таблицы

```
struct sqlite3_vtab {
    const sqlite3_module *pModule; /* модуль таблицы */
    int nRef; /* число ссылок, инициализирующееся ядром SQLite */
    char *zErrMsg; /* для передачи сообщений об ошибках ядру */
};
```

Методы прохода по записям таблицы, использующие для этого структуру курсора `sqlite3_vtab_cursor`, представленую на листинге 1.6, над которой также реализуют обертку для хранения необходимых для обработки переменных.

Листинг 1.6 — Структура курсора

```
struct sqlite3_vtab_cursor {
  sqlite3_vtab *pVtab; /* указатель на виртуальную таблицу */
};
```

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 1.7.

Листинг 1.7 — Пользовательские структуры виртуальной таблицы и курсора

```
typedef struct lindex_vtab {
    sqlite3_vtab base;  /* основа виртуальной таблицы */
    sqlite3 *db;        /* подключение к базе данных */
    sqlite3_stmt *stmt; /* инструкция доступа к записи по ROWID*/
    PyObject *lindex;   /* собственно объект индекса */
} lindex_vtab;

typedef struct lindex_cursor {
    sqlite3_vtab_cursor base; /* базовая структура курсора */
    PyObject *rowids;         /* массив выбранных ROWID */
    PyArrayIterObject *iter;  /* итератор по массиву ROWID */
} lindex_cursor;
```

На листинге 1.8 представлена реализация метода создания индекса, реализованного в качестве `xCreate`. Код инициализации и запуска обучение индекса в Python через программиный интерфейс приведен на листинге 1.9.

Листинг 1.8 — Создание индекса

```
int lindexCreate(sqlite3 *db, void *pAux,
                 const int argc, const char *const *argv,
                 sqlite3_vtab **ppVtab, char **errMsg)
{
    lindex_vtab *vtab = sqlite3_malloc(sizeof(lindex_vtab));

    memset(vtab, 0, sizeof(*vtab));
    *ppVtab = &vtab->base;

    const char *vTableName = argv[2];
    const char *rTableName = vTableName + 4;

    char *sql_template = "SELECT sql FROM sqlite_master WHERE
    ↪  type='table' AND name='%s';";
```

Продолжение листинга 1.8

```
    char *schemaQuery = sqlite3_mprintf(sql_template, rTableName);

    char* messaggeError;
    char vSqlQuery[10000];
    strcpy(vSqlQuery, vTableName);
    sqlite3_exec(db, schemaQuery, callback, vSqlQuery, &messaggeError);
    char *resVSqlQuery = sqlite3_mprintf("%s;", vSqlQuery);

    sqlite3_declare_vtab(db, resVSqlQuery);

    sqlite3_free(schemaQuery);
    sqlite3_free(resVSqlQuery);

    long column_index = strtol(argv[3], NULL, 10);
    const char *model = argv[4];

    initPythonIndex(db, rTableName, model, column_index, vtab);

    char* result_query = sqlite3_mprintf("SELECT * FROM %s WHERE ROWID =
    ↪  ?;", rTableName);
    sqlite3_prepare_v2(db, result_query, -1, &vtab->stmt, NULL);

    return SQLITE_OK;
}
```

Листинг 1.9 — Инициализация индекса

```
int initPythonIndex(sqlite3 *db,
                    const char *const tableName,
                    const char *const modelName,
                    const long column_index,
                    lindex_vtab *vTab) {
    char* query = sqlite3_mprintf("SELECT ROWID, * FROM %s", tableName);

    sqlite3_stmt* stmt;
    sqlite3_prepare_v2(db, query, -1, &stmt, NULL);
    sqlite3_free(query);

    PyObject* builderModule = PyImport_ImportModule("indexes.builder");

    PyObject* builderClassName = PyObject_GetAttrString(builderModule,
    ↪  "LindexBuilder");
    PyObject* pyModelName = PyTuple_Pack(1,
    ↪  PyUnicode_FromString(modelName));
```

Продолжение листинга 1.9

```
16      PyObject* builder = PyObject_CallObject(builderClassName,
        ↪  pyModelName);
17
18      PyObject* lindex = PyObject_CallMethod(builder, "build", NULL);
19      PyObject* keys = PyList_New(0);
20      PyObject* rows = PyList_New(0);
21
22      int i = 0;
23      while (sqlite3_step(stmt) == SQLITE_ROW) {
24          int64_t key = sqlite3_column_int64(stmt, column_index);
25          int64_t rowid = sqlite3_column_int64(stmt, 0);
26
27          PyList_Append(keys, PyLong_FromLong(key));
28          PyList_Append(rows, PyLong_FromLong(rowid));
29
30          i++;
31      }
32
33      if (i) {
34          PyObject* train = PyUnicode_FromString("train");
35          PyObject* check = PyObject_CallMethodObjArgs(lindex, train,
            ↪  keys, rows, NULL);
36      }
37
38      vTab->lindex = lindex;
39      vTab->db = db;
40
41      Py_DECREF(keys);
42      /* ... */
43      sqlite3_finalize(stmt);
44
45      return SQLITE_OK;
46  }
```

Обработка параметров фильтрации приведена, получение массива подходящий строк и проход для их вывода приведены в приложении А на листингах А.1, А.2 и А.3 соответственно.

Реализация вставки в виртуальную таблицу приведена на листинге 1.10.

Листинг 1.10 — Реализация вставки в виртуальную таблицу

```
1  int lindexUpdate(sqlite3_vtab *pVTab,
2                   int argc, sqlite3_value **argv,
3                   sqlite_int64 *pRowid) {
```

Продолжение листинга 1.10

```
    if (!(argc > 1 && sqlite3_value_type(argv[0]) == SQLITE_NULL))
        return SQLITE_CONSTRAINT;

    lindex_vtab *lTab = (lindex_vtab*)pVTab;
    sqlite3 *db = lTab->db;
    int64_t column = sqlite3_value_int64(argv[2]);
    char *query = sqlite3_mprintf("INSERT INTO maps VALUES(%d);",
    ↪  column);
    sqlite3_exec(db, query, 0, 0, 0);
    sqlite3_int64 lastRowID = sqlite3_last_insert_rowid(db);

    PyObject* insert = PyUnicode_FromString("insert");
    PyObject* key = PyLong_FromLongLong(column);
    PyObject* data = PyLong_FromLongLong(lastRowID);

    PyObject_CallMethodObjArgs(lTab->lindex, insert, key, data, NULL);

    return SQLITE_OK;
}
```

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 1.11.

Листинг 1.11 — Ключевые моменты сборки программного обеспечения

```
PATHFLAG := -I$(INCDIR)
CFLAGS := -std=c99 $(PATHFLAG) -fPIC
ADD_LIBS :=  -lsqlite3 -lpython3.10
```

Продолжение листинга 1.11

```
4  PYTHON_PATH:= $(shell pkg-config --cflags --libs python3)
5  NUMPY_PATH := -I$(shell pip show numpy
6                             | grep -oP "(?<=Location: ).*"$\
7                             | awk '{$$1=$$1};1')/numpy/core/include
8
9  SRCS := $(wildcard $(SRCDIR)*.c)
10 OBGS := $(patsubst $(SRCDIR)%.c,$(OUTDIR)%.o,$(SRCS))
11
12 .PHONY : clean build
13
14 build: lindex.so
15
16 %.so : $(OBGS)
17         @mkdir -p $(@D)
18         $(CC) $(ADD_LIBS) -shared $^ -o $@
19
20 $(OUTDIR)%.o : %.c
21         @mkdir -p $(@D)
22         $(CC) $(CFLAGS) $(PYTHON_PATH) $(NUMPY_PATH) -c $< -o $@
```

Для работы компонента индекса, реализованного на Python, требуется установка зависимостей из уже сформированного файла `requirements.txt` путем выполения команды, представленной на листинге 1.12. Также для штатной работы программного обеспечения требуется прописать путь к модулям, реализованным на языке Python, что приведено на том же листинге.

Листинг 1.12 — Подготовка для работы модулей Python

```
1 $ pip install -r requirements.txt
2 $ export PYTHONPATH=<путь_к_модулям>:$PYTHONPATH
```

1.4 Взаимодействие с программным обеспечением

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 1.13

Листинг 1.13 — 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 тестирование программного обеспечения, взаимодействие с которым происходит через командную строку sqlite3. Пример работы и тестирования представлен на листинге 1.14.

Листинг 1.14 — Пример работы программного обеспечения

```
$ sqlite3 test.db
sqlite> .load ./lindex
sqlite> create table maps(keys INTEGER);
sqlite> .mode csv
sqlite> .import osm100000.csv maps;
sqlite> create virtual table virtmaps using lindex(0, "FCNN2");
Epoch 1/30
1001/1001 [====================] - 1s 768us/step - loss: 0.0011
...
sqlite> select * from virtmaps where keys = 232131750;
keys
232131750
```

ПРИЛОЖЕНИЕ А

Реализация курсора виртуальной таблицы

Листинг А.1 — Обработка параметров фильтрации запроса

```
int lindexBestIndex(sqlite3_vtab *tab,
                    sqlite3_index_info *pIndexInfo)
{
    if (pIndexInfo->nConstraint == 1)
    {
        if (pIndexInfo->aConstraint[0].usable)
        {
            int mode;
            switch (pIndexInfo->aConstraint[0].op)
            {
                case SQLITE_INDEX_CONSTRAINT_EQ:
                    mode = 0;
                    break;
                case SQLITE_INDEX_CONSTRAINT_GT:
                    mode = 11;
                    break;
                case SQLITE_INDEX_CONSTRAINT_GE:
                    mode = 10;
                    break;
                case SQLITE_INDEX_CONSTRAINT_LT:
                    mode = 21;
                    break;
                case SQLITE_INDEX_CONSTRAINT_LE:
                    mode = 20;
                    break;
                default:
                    return SQLITE_CONSTRAINT;
            }
            pIndexInfo->idxNum = mode;
            pIndexInfo->aConstraintUsage[0].argvIndex = 1;
            pIndexInfo->aConstraintUsage[0].omit = 1;

            return SQLITE_OK;
        }

        return SQLITE_CONSTRAINT;
    }

    if (pIndexInfo->nConstraint == 2)
    {
        for (int i = 0; i < pIndexInfo->nConstraint; i++)
```

Продолжение листинга А.1

```
        {
            switch (pIndexInfo->aConstraint[i].op)
            {
                case SQLITE_INDEX_CONSTRAINT_LE:
                    break;
                case SQLITE_INDEX_CONSTRAINT_GE:
                    break;
                default:
                    return SQLITE_CONSTRAINT;
            }

            pIndexInfo->aConstraintUsage[i].argvIndex = i + 1;
            pIndexInfo->aConstraintUsage[i].omit = 1;
        }

        pIndexInfo->idxNum = 30;

        return SQLITE_OK;
    }

    return SQLITE_CONSTRAINT;
}
```

Листинг А.2 — Выбор строк, удовлетворяющих фильтру

```
int lindexFilter(sqlite3_vtab_cursor *cur, int idxNum,
                 const char *idxStr, int argc,
                 sqlite3_value **argv) {
    import_array()
    lindex_vtab *lTab = (lindex_vtab*)cur->pVtab;

    PyObject* keys = PyList_New(0);

    for (int i = 0; i < argc; ++i) {
        int64_t value = (int64_t)sqlite3_value_int64(argv[i]);
        PyList_Append(keys, PyLong_FromLong(value));
    }

    PyObject* tuple_rowids;

    if (!idxNum) {
        PyObject* find = PyUnicode_FromString("find");
        tuple_rowids = PyObject_CallMethodObjArgs(lTab->lindex, find,
        ↪  keys, NULL);
    }
```

Продолжение листинга А.2

```
    else {
        PyObject* constraints = PyList_New(0);
        PyObject* noneObj = Py_None;

        for (int i = 0; i < argc; ++i) {
            PyList_Append(constraints, PyLong_FromLong(idxNum % 10));
        }

        if (idxNum / 10 != 3) {
            Py_INCREF(noneObj);
            PyList_Insert(keys, idxNum / 10 % 2, noneObj);

            Py_INCREF(noneObj);
            PyList_Insert(constraints, idxNum / 10 % 2, noneObj);
        }

        PyObject* prange= PyUnicode_FromString("predict_range");
        tuple_rowids = PyObject_CallMethodObjArgs(lTab->lindex, prange,
                keys, constraints, NULL);
    }

    PyObject* rowids;

    int tmp;
    PyArg_ParseTuple(tuple_rowids, "Oi", &rowids, &tmp);
    npy_intp size = PyArray_SIZE(rowids);
    PyArrayIterObject *iter = (PyArrayIterObject
    ↪  *)PyArray_IterNew(rowids);

    lindex_cursor *pCur = (lindex_cursor*)cur;
    pCur->rowids = rowids;
    pCur->iter = iter;

    int64_t rowid = *(int64_t *)PyArray_ITER_DATA(pCur->iter);
    sqlite3_bind_int64(lTab->stmt, 1, rowid);
    sqlite3_step(lTab->stmt);

    return SQLITE_OK;
}
```

Листинг А.3 — Реализация работы курсора

```
int lindexNext(sqlite3_vtab_cursor *cur) {
    lindex_cursor *pCur = (lindex_cursor*)cur;
    lindex_vtab *lTab = (lindex_vtab*)cur->pVtab;
```

Продолжение листинга А.3

```
4
5       PyArray_ITER_NEXT(pCur->iter);
6
7       sqlite3_reset(lTab->stmt);
8       sqlite3_clear_bindings(lTab->stmt);
9
10      int64_t rowid = *(int64_t *)PyArray_ITER_DATA(pCur->iter);
11      sqlite3_bind_int64(lTab->stmt, 1, rowid);
12      sqlite3_step(lTab->stmt);
13
14      return SQLITE_OK;
15  }
16
17  int lindexColumn(sqlite3_vtab_cursor *cur,
18                   sqlite3_context *ctx,
19                   int i)
20  {
21      lindex_vtab *lTab = (lindex_vtab*)cur->pVtab;
22      int64_t columnValue = sqlite3_column_int64(lTab->stmt, i);
23      sqlite3_result_int64(ctx, columnValue);
24
25      return SQLITE_OK;
26  }
27
28  int lindexEof(sqlite3_vtab_cursor *cur)
29  {
30      lindex_cursor *pCur = (lindex_cursor*)cur;
31      return !PyArray_ITER_NOTDONE(pCur->iter);
32  }
```