Advanced SQL c SQLite

- Работа с SQL через API
- python
- C/C++
- Embedded SQL

Простейшая программа на C

```c
#include <sqlite3.h>
#include <stdio.h>

int main(void){
    printf("%s\n", sqlite3_libversion());
    return 0;
}
```

Как компилировать в Linux

Получаем пакет

```
sudo apt-get install libsqlite3-dev
```

Компилируем код

```
gcc -o sqlite3_ver sqlite3_ver.c -lsqlite3 -std=c99
```

Как компилировать в Windows

Работа с запросами ч.1

Инициализируем переменные

```
#include <sqlite3.h>
#include <stdio.h>

int main(void){
    sqlite3 *db;
    sqlite3_stmt *res;
    int rc = sqlite3_open(":memory:", &db);
```

sqlite3 - структура для работы с БД

sqlite3_stmt - структура для обработки объектов

Жизненный цикл выполнения SQL кода в sqlite

1. Создать объект для подготовительного выражения. используя функцию sqlite3_prepare_v2()
2. Присвоить значения для параметров, используя sqlite3_bind*_() интерфейсы
3. Выполнить SQL код через функцию sqlite3_step() один или несколько раз
4. Обновить подготовительное выражение, используя sqlite3_reset() и вернуться к шагу 2. Выполнить один или несколько раз
5. Уничтожить объект, используя sqlite3_finalize()

sqlite3_open

```
int sqlite3_open(
  const char *filename,   /* Database filename (UTF-8) */
  sqlite3 **ppDb          /* OUT: SQLite db handle */
);
int sqlite3_open16(
  const void *filename,   /* Database filename (UTF-16) */
  sqlite3 **ppDb          /* OUT: SQLite db handle */
);
```

Открытие БД, указанное в файле. передаваемое параметром filename. В объект ppDb возвращается указатель на драйвер БД.

sqlite3_open_v2

```
int sqlite3_open_v2(
  const char *filename,   /* Database filename (UTF-8) */
  sqlite3 **ppDb,         /* OUT: SQLite db handle */
  int flags,              /* Flags */
  const char *zVfs        /* Name of VFS module to use */
);
```

flags - 3 стандартных варианта (SQLITE_OPEN_READONLY, SQLITE_OPEN_READWRITE, SQLITE_OPEN_READWRITE | SQLITE_OPEN_CREATE) + дополнительные для работы в multi-thread режиме

VFS - специальный модуль для портабельности

Работа с запросами ч. 2

```
  rc = sqlite3_prepare_v2(db, "SELECT SQLITE_VERSION()",
                          -1, &res, 0);
    if (rc != SQLITE_OK) {

        fprintf(stderr, "Failed to fetch data: %s\n",
                          sqlite3_errmsg(db));
        sqlite3_close(db);

        return 1;
    }
```

```
rc = sqlite3_prepare_v2(db, "SELECT SQLITE_VERSION()",
                        -1, &res, 0);
```

```
int sqlite3_prepare_v2(
  sqlite3 *db,            /* Database handle */
  const char *zSql,       /* SQL statement, UTF-8 encoded
  int nByte,              /* Maximum length of zSql in byt
  sqlite3_stmt **ppStmt,  /* OUT: Statement handle */
  const char **pzTail     /* OUT: Pointer to unused portic
);
```

*db - соединение с БД

*zSql - выражение, которое должно быть скомпилированно

nByte - -1, когда происходит чтение до первого нулевого терминатора, позитивное число - количество байт для чтения

**pzTail - первый байт идущий за выражением в *zSql

**ppStmt - выражение, которое будет скомпилировано после sqlite3_step()

```
if (rc != SQLITE_OK) {
        fprintf(stderr, "Failed to fetch data: %s\n",
                sqlite3_errmsg(db));
    sqlite3_close(db);
    return 1;
}
```

SQLITE_OK - результирующий код. означающий, что выражение отработало корректно

```
if (rc != SQLITE_OK) {
        fprintf(stderr, "Failed to fetch data: %s\n",
                sqlite3_errmsg(db));
    sqlite3_close(db);
    return 1;
}
```

```
const char *sqlite3_errmsg(sqlite3*);
```

sqlite3_errmsg() возвращает текст, описывающий ошибку.

Память для управления строкой ошибки управляется вручную.

Работа с запросам ч.3

```
    rc = sqlite3_step(res);
    if (rc == SQLITE_ROW){
        printf("%s\n", sqlite3_column_text(res, 0));
    }

    sqlite3_finalize(res);
    sqlite3_close(db);
```

sqlite3_step()

```
int sqlite3_step(sqlite3_stmt*);
```

Обработка выражения. Возможные варианта возврата - SQLITE_BUSY, SQLITE_DONE, SQLITE_ROW, SQLITE_ERROR, или SQLITE_MISUSE и т. д.

SQLITE_BUSY - индикатор того, что файл БД на данный момент не может быть записан из-за другого соединения к БД из под другого соединения.

SQLITE_DONE - индикатор того, что операция была выполнена. Обычно появляется, когда выборка SQL выражения окончена.

sqlite3_step()

```
int sqlite3_step(sqlite3_stmt*);
```

SQLITE_ROW - когда вызов SQL-выражения вернул какие-либо данные.

SQLITE_ERROR - ошибка во время runtime выполнения. Описание ошибки можно узнать, вызвав sqlite3_errmsg().

SQLITE_MISUSE - указание того, что sqlite3_step вызвана некорректно (не в тот момент, например, что все данные уже были считаны).