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).

Python

Введение в язык программирования

Лабораторная

Python

Работа с СУБД

- **sqlite3**
- **sqlalchemy**

Python соединение с БД

```
import sqlite3
db = sqlite3.connect('chinook.db')
```

Объект класса Connection

Python Connection class

Методы и атрибута

- isolation level - переменная, отвечающая за тип изоляции
- cursor(factory=Cursor) - метод для вызова cursor. Возвращает объект типа Cursor.
- commit() - завершение текущей транзакции
- rollback() - откат текущей транзакции
- execute(sql[, parameters]) - создание промежуточного объекта типа cursor и вызов метода execute объекта cursor
- executemany(sql[, parameters])- создание промежуточного объекта типа cursor и вызов метода cursor объекта executemany

Python3 создание объекта типа cursor

```
import sqlite3
db = sqlite3.connect('chinook.db')
cursor=db.cursor()
cursor.execute('''
    DROP TABLE IF EXISTS students''')
```

Объект класса Cursor

Python3 class Cursor

методы и атрибуты

- execute(sql[, parameters]) - вызов SQL команды
- executemany(sql, seq_of_parameters) - вызов SQL команды для всех последовательностей команд
- executescript(sql_script) - вызов нескольких SQL-выражений за раз.
- fetchone() - получение следующей строки данных или тип None, если объект не найден
- fetchall() - получение всех оставшихся строк, результат список

Python 3

Лабораторная

Advanced SQL

- выражение WITH
- Оконные функции
- Процедуры и функции

Выражение WITH

Common Table Expression (CTE)

Данные выражения позволяют "запоминать" какой-то SELECT в определенном имени и затем использовать его в дальнейшем

```
WITH A(id)
AS
(select 1)
SELECT *
  FROM A a1
  JOIN A a2
  ON a1.id = a2.id;
```

Рекурсивное выражение WITH

Рекурсивное CTE может быть использовано для обхода дерева или графа. Синтаксис похож на обычный CTE, однако, есть ряд дополнительных свойств:

- with выражение должно состоять из двух выражений, объединенных select
- объявленная таблица в with должна быть объявлена один раз справа

Рекурсивное выражение WITH

```
WITH RECURSIVE
  cnt(x) AS (VALUES(1) UNION ALL SELECT x+1 FROM cnt WHERE
SELECT x FROM cnt;
```

```
WITH RECURSIVE
  cnt(x) AS (
     SELECT 1
     UNION ALL
     SELECT x+1 FROM cnt
      LIMIT 1000000
  )
SELECT x FROM cnt;
```

Рекурсивное выражение WITH

Рассмотрим таблицу участников организации

```
CREATE TABLE org(
  name TEXT PRIMARY KEY,
  boss TEXT REFERENCES org,
  height INT,
  -- other content omitted
);
```

У каждого члена организации есть имя, и почти у всех есть босс (у главы организации поле boss - NULL)

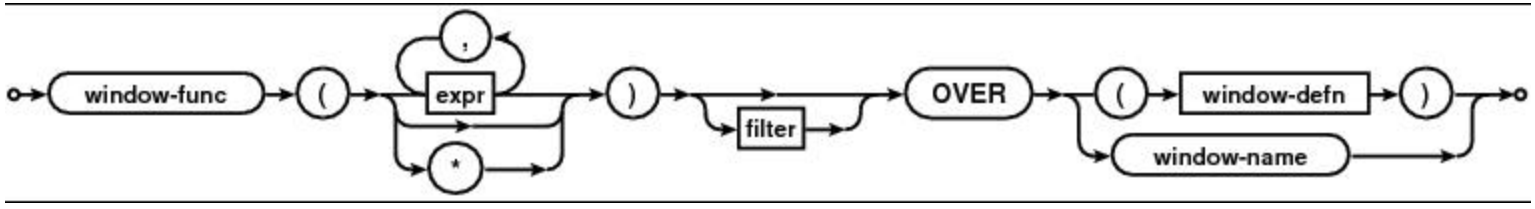
Рекурсивное выражение WITH

Следующий запрос показывает средний рост всех менеджеров компании, которык являются менеджерами Alice (включая саму Alice)

```
WITH RECURSIVE
  works_for_alice(n) AS (
    VALUES('Alice')
    UNION
    SELECT name FROM org, works_for_alice
     WHERE org.boss=works_for_alice.n
  )
SELECT avg(height) FROM org
 WHERE org.name IN works_for_alice;
```

Оконные функции

Оконная функция - специальная SQL функция, где входящие значения берутся из "окна" одной или более строк в результирующее выражение выражения SELECT.



Оконные функции отличаются от обычных SQL-функций существованием предложением OVER()

Оконные функции

Рассмотрим таблицу

```
CREATE TABLE t0(x INTEGER PRIMARY KEY, y TEXT);
INSERT INTO t0 VALUES (1, 'aaa'), (2, 'ccc'), (3, 'bbb');
```

Пример оконной функции:

```
SELECT x, y, row_number() OVER (ORDER BY y) AS row_number
```

В данном случае используется встроенная оконная функция row_number(). Функция row_number() присваивает монотонно увеличивающееся целое число для каждого строки в возникающем окне.

Оконные функции. Пример другого синтаксиса.

```
SELECT x, y, row_number() OVER win1, rank() OVER win2
FROM t0
WINDOW win1 AS (ORDER BY y RANGE BETWEEN UNBOUNDED PRECEDI
       win2 AS (PARTITION BY y ORDER BY x)
ORDER BY x;
```

Агрегатные оконные функции

Рассмотрим таблицу

```
CREATE TABLE t1(a INTEGER PRIMARY KEY, b, c);
INSERT INTO t1 VALUES   (1, 'A', 'one'  ),
                        (2, 'B', 'two'  ),
                        (3, 'C', 'three'),
                        (4, 'D', 'one'  ),
                        (5, 'E', 'two'  ),
                        (6, 'F', 'three'),
                        (7, 'G', 'one'  );
```

Агрегатная оконная функция аналогична агрегатной функции, кроме того, что добавления запроса не изменяет количество строк. Для каждой строки фактически формируется своя группа (окно).

Агрегатные оконные функции

```
SELECT a, b, group_concat(b, '.') OVER (
  ORDER BY a ROWS BETWEEN 1 PRECEDING AND 1 FOLLOWING
) AS group_concat FROM t1;
```

В данном примере окном являются три строки: текущая строка, следующая строка, предыдущая строка. После сортировки в OVER(), экземпляр окна состоит из всех строк, которые подходят для сортировки относительно текущего значения. При этом, если какие-то значения возвращают то же самое значение в результате сортировки, то они всегда будут в окне либо все вместе, либо ни одного.

```
SELECT a, b, c,
       group_concat(b, '.') OVER (ORDER BY c) AS group_cor
FROM t1 ORDER BY a;
```

Спецификация окна (frame-spec)

frame-spec определяет. какие строки считываются агрегатной оконной функцией. Frame-spec состои из 3 частей:

- тип окна - или Range, или Rows
- начальная граница окна
- конечная граница окна

Конечная граница окна может быть опущена, в таком случае по умолчанию используется CURRENT ROW. Спецификация окна по умолчанию:

```
RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW
```

Значение по умолчанию означает, что агрегатная оконная функция считывает все строки по умолчанию от начала раздела до текущей строки и совпадающей с ней в рамках сортировки (peer)

Range или ROW окна

Если тип окна RANGE, то строки с одинаковыми значениями для всего ORDER BY выражения называются равноправными (peers). Если не указано ORDER BY выражение, то все строки равноправны. Равноправные строки всегда имеют одинаковые окна.

Границы окон

Есть пять вариантов для границ окна:

- **UNBOUNDED PRECEDING** - начало окна - первая строка в множестве
- **<expr> PRECEDING** - константное значение. Используется только в ROWS окнах. Сколько строк до.
- **CURRENT ROW** - текущая строка. Все peers также включены в окно
- **<expr> FOLLOWING** - константное значение. Используется только в ROWS окнах. Сколько строк после.
- **UNBOUNDED FOLLOWING** - конец окна - первая строка в множестве

Границы окон

Граница сверху всегда должна быть выше, чем граница снизу

```
SELECT c, a, b, group_concat(b, '.') OVER (
  ORDER BY c, a ROWS BETWEEN CURRENT ROW AND UNBOUNDED FOI
) AS group_concat
FROM t1 ORDER BY c, a;
```

Выражение PARTITION BY

Определение окна может включать в себя выражение PARTITION BY. В таком случае все строки делятся на группы (партиции) с одними и теми же значениями в выражении PARTITION BY, и тогда оконная функция выполняется раздельно для каждой партиции.

```
SELECT c, a, b, group_concat(b, '.') OVER (
  PARTITION BY c ORDER BY a RANGE BETWEEN CURRENT ROW AND
) AS group_concat
FROM t1 ORDER BY c, a;
```

Выражение FILTER

Выражение FILTER используется для того, чтобы осуществлять фильтр на окне. При этом агрегатная функция все еще возвращает значение для каждой строки, но те строки, для которых вычисление FILTER не равно true, не включается в окно.

```
SELECT c, a, b, group_concat(b, '.') FILTER (WHERE c!='two
  ORDER BY a
) AS group_concat
FROM t1 ORDER BY a;
```

Агрегатные оконные функции

- min
- max
- count
- sum
- avg
- group_concat

Встроенные оконные функции

Встроенные оконные функции используют PARTITIOB BY выражение, но при этом в большей части игнорируют границы окон (кроме first_value(), last_value(), nth_value())

- row_number() - нумерование строк
- rank() - rank текущей строки с пробелами
- dense_rank() - rank без пропусков
- percent_rank() - (rank-1)/(partition_rows-1)
- cume_dist() - row-number/partition-rows
- ntile(N) - N-integer. разделение партиции на N групп, порядок определяется в ORDER BY.

Встроенные оконные функции

- lag(expr)
- lag(expr, offset)
- lag(expr, offset, default) - предыдущее значение в рамках партиции
- lead(expr)
- lead(expr, offset)
- lead(expr, offset, default) - следующее значение в рамках партиции

Встроенные оконные функции

- first_value(expr) - первое значение в рамках окна
- last_value(expr) - последнее значение в рамках окна
- nth_value(expr, N) - N-ое значение в рамках окна.