Национальный исследовательский университет
“Московский энергетический институт”

**Отчет по курсовой работе**

**по дисциплине: «Программная инженерия»**

**Тема курсовой работы**

**«**Создание базы данных SQLite и работа с ней на Swift (Xcode)**»**

Группа: А-05-18

Студентка: Поиленкова А.А

Преподаватель: Маран М.М

Москва 2020

# Постановка задачи

Разработать IOS-приложение с подключением базы данных (программный доступ).

БД: SQLite
Среда: Xcode ( язык Swift )

Визуальное моделирование выполнено в программе MySQL Workbench

# Идея

Приложение будет содержать информацию о напитках и часах работы кофеен.
С помощью задания параметров мы будем получать места (адреса кофеен), которые удовлетворят требованию.

Примеры запросов:

- Вывести список всех мест
- Где купить капучино до 200 рублей?
- Где купить раф до 100 рублей?

Ответом будет выведен список кофеен, содержащий имя, адрес и часы работы.

# Ход решения, поставленной задачи

БД

1. Проектирование базы данных
2. Создание БД
3. Заполнение БД
4. Проверка работы на корректность

Разработка приложения

5. Установление соединения с БД
6. Прорисовка внешнего вида создание внешних компонентов
7. Создание внешних компонентов с их связями и дополнительных классов
8. Написание кода для каждого окна
9. Тестирование

# Базы данных

SQLite — это встраиваемая кросс-платформенная БД, которая поддерживает достаточно полный набор команд SQL. Относится к реляционным системам управления базами данных.

## Проектирование базы данных

Важным этапом работы с БД является проектирование. Хорошо спроектированная модель облегчит работу в дальнейшем, потому что устройство БД напрямую влияет на запросы, которые нужно будет выполнять для получения данных.

Проектирование решает два вопроса:
1. Что храним?
2. С какими связями?

Таблицы:

На этапе проектирование мы задаем имя таблицы и количество колон Для каждой колоны мы задаем имя, тип и выставляем флаги

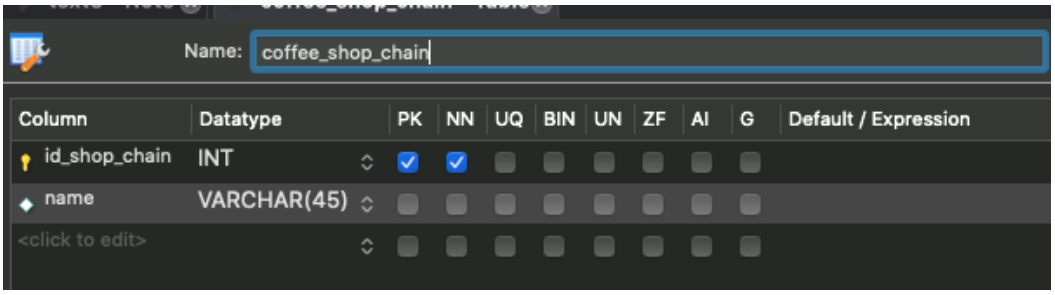



Таблица расшифровки флагов:

| | |
|---|---|
| **PK** | первичный ключ |
| **NN** | Not Null |
| **BIN** | Binary (хранит данные как двоичные строки) |
| **UN** | Без знака (диапазон будет таким же, но сместится и будет начинается с 0) |
| **UQ** | Создать/удалить уникальный ключ |
| **ZF** | Zero-Filled (если длина равна 5, как INT (5), то каждое поле заполняется 0s до 5-го значения. 12 = 00012, 400 = 00400 и т.д.) |
| **G** | сформированный столбец формулой, основанной на других столбцах |
| **AI** | автоматический приращение |

**Ключ**
- атрибут, который позволяет индексировать запись ( обычно INT )

Суперключ - атрибут или набор атрибутов, позволяющий уникально индексировать запись

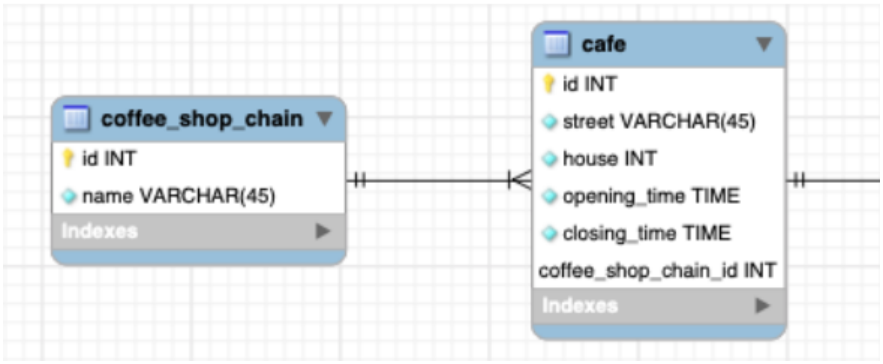
Потенциальный ключ - обладает свойством минимальности. Суперключ внутри которого нет другого суперключа

Первичный ключ - потенциальный ключ, выбранный для однозначной идентификации отношения

Вторичный ключ - любой другой ключ, кроме первичного

Внешний ключ - атрибут или множество атрибутов отношения, соответствующие потенциальному ключу некоторого отношения.
( обычно служат для связей между таблицами )

Таблицы связываем с помощью *внешних ключей*
Рассмотрим это на примере таблицы coffee_shop_chain и cafe:




Значение поля **id_shop_chain** (первичный ключ) из таблицы coffee_shop_chain копируется в таблицу cafe при вставке каждой записи. С помощью такого механизма каждое кафе привязано к какой-то кофейной сети ( связь - один ко многим ).

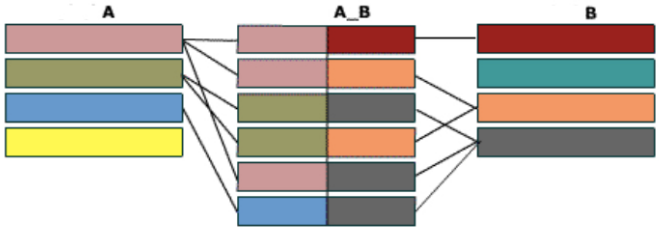
**Виды связей:**

1. Один к одному
2. Один ко многим
3. Многие ко многим

Наиболее сложной является связь «многие ко многим»
Напрямую соединить две таблицы ( A и B ) таким способом мы не сможем, поэтому будем добавлять соединительную таблицу (A_B), которая будет состоять из двух колон:

1 - внешний ключ из A

2 - внешний ключ из B




## Создание диаграммы сущность-связь
**(entity-relationship diagram, ERD)**

*Нормальные формы* – это рекомендации по проектированию баз данных

### 1. Первая нормальная форма

Таблица базы данных – это представление **сущности** вашей системы, которую вы создаете

- каждая таблица имеет первичный ключ, состоящий из наименьшего возможного количества полей.
- поля не имеют дубликатов в каждой записи и каждое поле содержит только одно значение
- порядок записей таблицы не должен иметь значения.

### 2. Вторая нормальная форма

- БД приведена к первой нормальной форме
- поля с не первичным ключом не должны быть зависимы от первичного ключа

### 3. Третья нормальная форма

- БД приведена ко второй нормальной форме
- не может быть транзитивных зависимостей между полями в таблице.

**Используемые типы данных**:

| Смысл | Тип |
|---|---|
| Ключи | **INT** |
| Цена и объем | **INT** с флагом UN ( тк по смыслу не могут быть отрицательными ) |
| Время открытия и закрытия | **TIME** |
| Названия | **VARCHAR** (стока переменной длины ) |

**Пояснение к БД:**

Для хранения названий сетей кофеен заведем таблицу coffee_shop_chain с полями имя и первичный ключ.

Следующая таблица(cafe) будет иметь более подробное описание каждой кофейни с фиксированным адресом, временем открытия и закрытия, а с помощью поля со внешним ключом будем привязывать их к сети (coffee_shop_chain)

Связь между таблицей coffee_shop_chain и cafe будет один ко многим, тк у одной сети может быть несколько зданий, где можно выпить кофе, но при этом кофейня может принадлежать только одной сети )

Напиток у нас состоит из следующих компонентов:

*Название + кофейная основа + база* + добавки*

*разные виды молока, вода, газировки

Поэтому создаем таблицу drink, которую свяжем с таблицами ( name_drink, coffee_content, base, additives )

Название, основа и база у оного напитка могут быть только одни, потому делаем связь один ко многим, а вот добавок может быть несколько (пр: корица и сироп), поэтому с добавками реализуем связь многие ко многим с помощью смежной таблицы (additives_drink)

Пример:

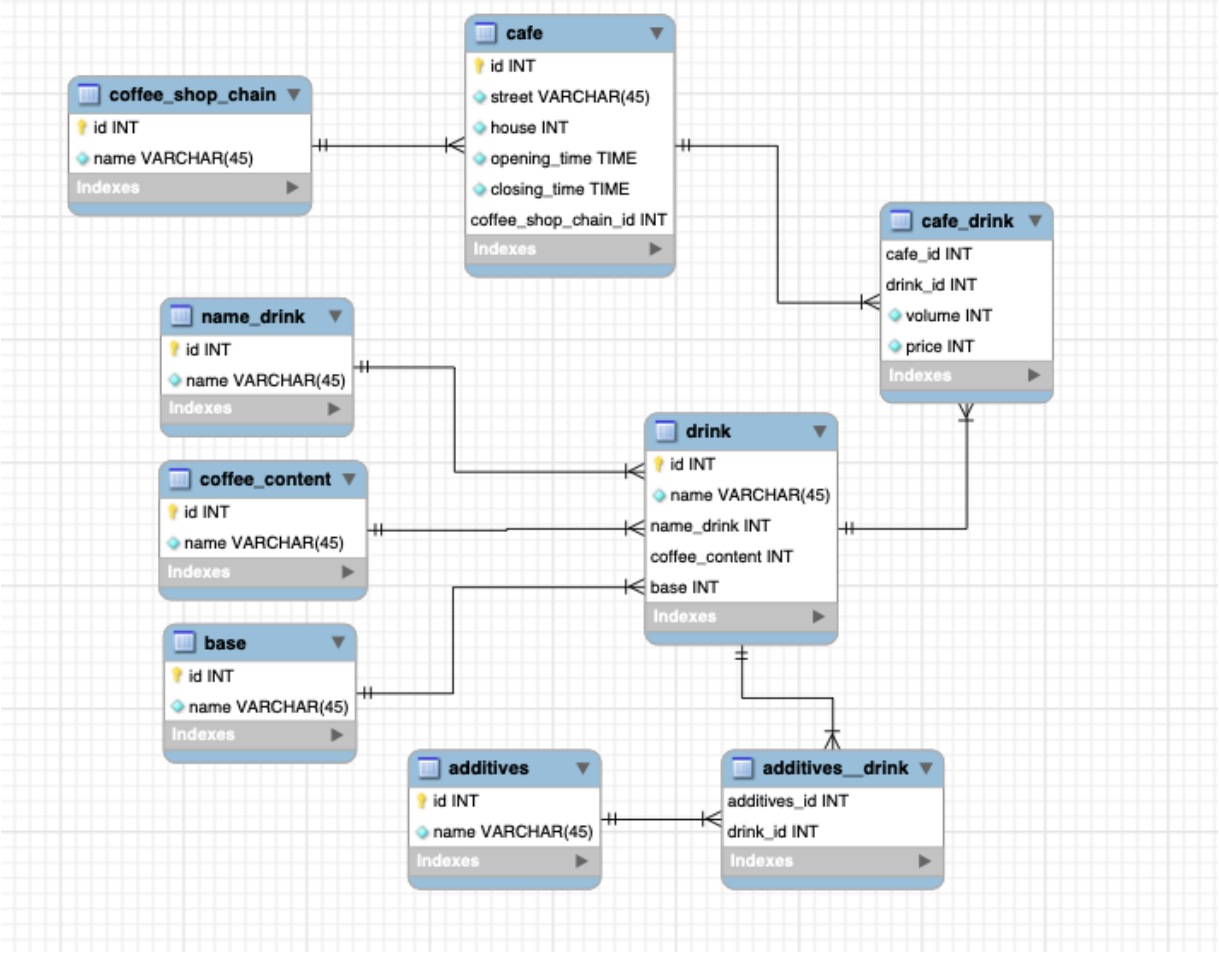
| name_drink | | | coffee_content | | | base | |
|---|---|---|---|---|---|---|---|
| **Name** | **Id** | | **Name** | **Id** | | **Name** | **Id** |
| Капучино | 1 | | Эфиопия | 1 | | Молоко | 1 |
| Латте | 2 | | Колумбия | 2 | | Вода | 2 |
| Эспрессо | 3 | | Ничего | 3 | | Ничего | 3 |

Напиток «капучино» у нас будет состоять из:
Название: Капучино (1)
Основа: Эфиопия (1)
База: Молоко (1)

Связь между таблицей напиток( drink) и кафе( cafe ) будет один ко многим, потому что у одной кофейни может быть много напитков, а один и тот же напиток может быть у нескольких заведений.

Все наши рассуждения оформим в таблицу с помощью MySQL Workbench

**Составим диаграмму для решающейся задачи:**



# Создание и заполнение БД

Будем взаимодействовать с базой данных через интерфейс командной строки sqlite3

Для начала работы
Набираем в Терминале
`sqlite3`

```
air-anna-2:Coffee anna$ sqlite3
SQLite version 3.28.0 2019-04-15 14:49:49
Enter ".help" for usage hints.
Connected to a transient in-memory database.
Use ".open FILENAME" to reopen on a persistent
 database.
```

- Создание файла(coffe.sqlite), который будет хранить нашу базу данных

`sqlite> .open coffe.sqlite`

- Следуя построенной диаграмме, создадим таблицы с помощью команды

`create table [имя] ( поля );`

- Заполняем таблицы данными

Пример работы на таблице name_drink

```
sqlite> create table name_drink(
   ...>     id INTEGER PRIMARY KEY AUTOINCREMENT,
   ...>     name text no null);
sqlite>
sqlite> insert into name_drink(name)
   ...> values ("капучино");
sqlite> insert into name_drink(name)
   ...> values ("латте");
sqlite> insert into name_drink(name)
   ...> values ("раф");
sqlite> insert into name_drink(name)
   ...> values ("V60");
sqlite> insert into name_drink(name)
   ...> values ("эспрессо");
sqlite> insert into name_drink(name)
   ...> values ("какао");
sqlite> insert into name_drink(name)
   ...> values ("фильтр");
sqlite> select * from name_dring
   ...> ;
Error: no such table: name_dring
sqlite> select * from name_drink;
1|капучино
2|латте
3|раф
4|V60
5|эспрессо
6|какао
7|фильтр
```

# Запросы SQL

SQL или Structured Query Language (язык структурированных запросов) — язык программирования, предназначенный для управления данными в СУБД. Все современные СУБД поддерживают SQL.

Будем получать данные из таблиц с помощью запросов

Общая структура запроса выглядит следующим образом:

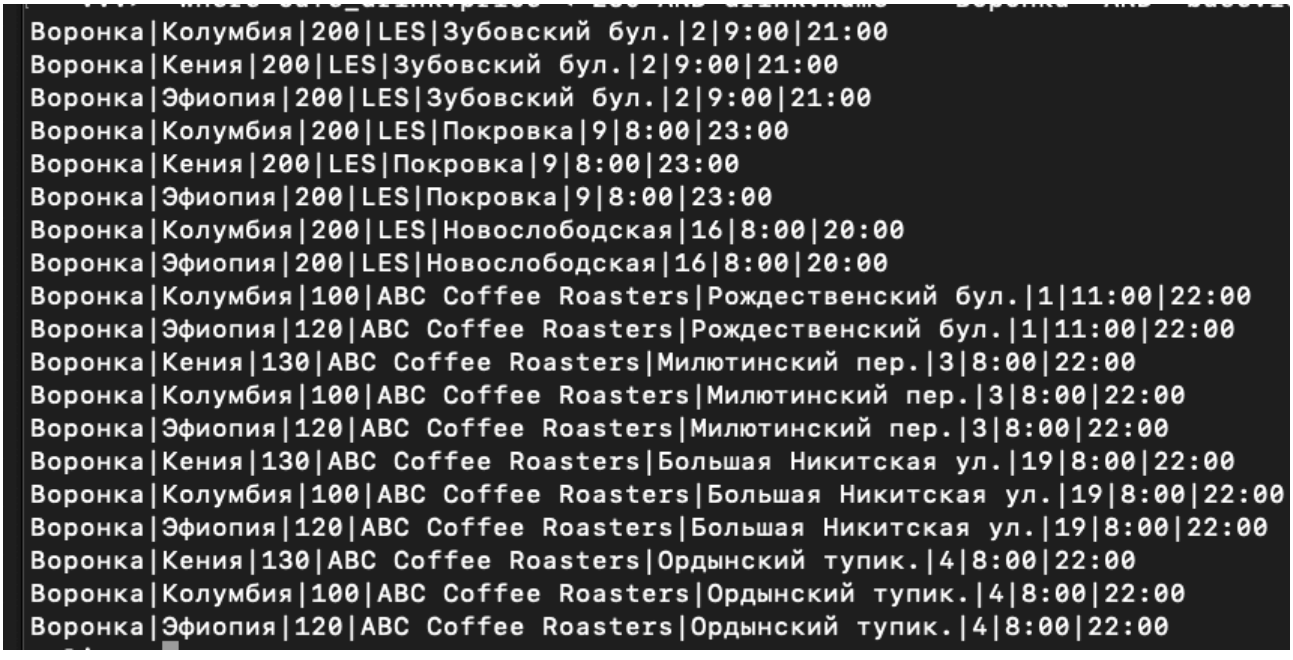
```
SELECT ('столбцы или * для выбора всех столбцов; обязательно')
FROM ('таблица; обязательно')
WHERE ('условие/фильтрация, например, city = 'Moscow'; необязательно')
GROUP BY ('столбец, по которому хотим сгруппировать данные; необязательно')
HAVING ('фильтрация на уровне сгруппированных данных; необязательно')
ORDER BY ('столбец, по которому хотим отсортировать вывод; необязательно')
```

Наша задача усложняется, потому что мы имеем несколько таблиц, из которых хотим получать данные, поэтому для объединения таблиц по ключу, который присутствует в обеих таблицах, будем использовать элемент JOIN.

Пример запроса, когда мы ходим получить черный кофе с названием Воронка и ценой ниже 250

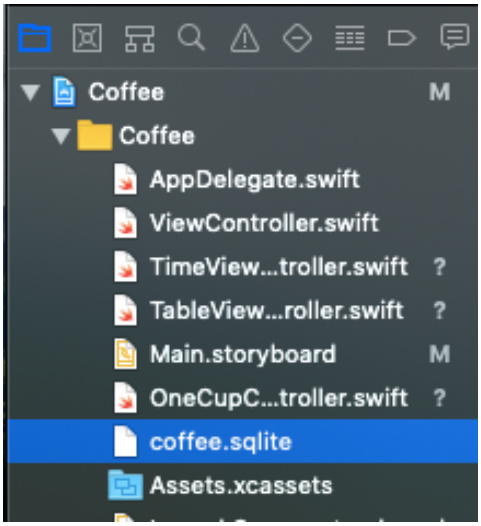
```
select drink.name, coffee_ content.name, cafe_drink.price,
coffee_shop_chain.name, cafe.street, cafe.house, cafe.opening_time,
cafe.closing_time
 from drink
 inner join cafe_drink ON drink_id = drink.id
 inner join base on base.id = drink.base_id
 inner join cafe ON cafe_id = cafe.id
 inner join coffee_content ON coffee_content_id = coffee_content.id
 inner join coffee_shop_chain ON coffee_shop_chain_id = coffee_shop_chain.id
 where cafe_drink.price < 250 AND drink.name = 'Воронка' AND  base.id IN ( 1, 6)
;
```

Получим:



# Программный доступ

В нашем проекте БД является встроенной и хранится на устройстве пользователя




**Плюсы:**

- Каждый пользователь сможет настраивать под себя базу данных
- Для использования не требуется подключение к интернету
- Работа происходит быстрее

**Минусы:**

- Работа с БД осуществляется только одним пользователем и на одном устройстве
- При удалении приложения БД также удаляется
- Нет возможности сделать резервную копию с накопленной информацией

## Установление соединения с БД:

- Добавляем файл БД(coffe.sqlite) в проект

- Добавляем библиотеку

```
import SQLite3
```

- Возвращает объект документа, инициализированный его местоположением в файловой системе. В нашем случае coffe.sqlite

Только для чтения, **String**.

```
let fileURL = URL.init(fileURLWithPath: Bundle.main.path(forResource: "coffee", ofType: «sqlite»)!)
```

*Bundle.main : получаем основной пакет приложения*

---

- Открываем файл БД с помощью sqlite3_open

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

Если база данных открыта (и / или создана) успешно, то возвращается SQLITE_OK. В противном случае возвращается код ошибки.

```
var db: OpaquePointer?   //указатель на объектс БД
```

**db** - это соединение с базой данных, полученное в результате предыдущего успешного вызова sqlite3_open().
Соединение с базой данных не должно быть закрыто.

```
if sqlite3_open(fileURL.path, &db) == SQLITE_OK
{
      StatusDB text = "connect"
}
```

Каждая открытая база данных SQLite представлена указателем на экземпляр структуры с именем "sqlite3".
Интерфейс sqlite3_open() - выполняет роль конструктора и создает объект подключения к БД
sqlite3_close() - выполняет роль деструктора и закрывает соединение с БД
sqlite3_prepare_v2() - функция, которая выполняет запрос к БД при помощи объекта подключения к базам данных.

- Выполнение запросов ( при успешном открытии БД ), используя функцию

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

Если **nByte < 0**, то **zSql** является чтение до первого нулевого символа.
Если **nByte > 0**, то это количество байтов, считанных из **zSql**.
Если **nByte = 0,** то подготовленная инструкция не генерируется.

***ppStmt** указывает на скомпилированный подготовленный оператор, который может быть выполнен с помощью sqlite3_step().
Если есть ошибка, то ***ppStmt** имеет значение NULL.

Если **pzTail** не равен NULL, то ***pzTail** должен указывать на первый байт после конца первого оператора SQL в **zSql**. Эти процедуры компилируют только первый оператор в **zSql**, поэтому ***pzTail** остается указывающим на то, что остается некомпилированным.

Аналогично sqlite3_open при успешном выполнении возвращает SQLITE_OK, а в противном код ошибки.

Пример выполнения запроса
select name from coffee_shop_chain; // вывод название сетей кофеен
И запись в массив **outTableCoffeeShop**

```
var outTableCoffeeShop: [String] = [] // объявляем в начале класса
```

---

```
var statement: OpaquePointer?

if sqlite3_prepare_v2(db, "select name from
coffee_shop_chain;", -1, &statement, nil) == SQLITE_OK {
                while sqlite3_step(statement) == SQLITE_ROW
                {
                    outTableCoffeeShop.append(String(cString:
sqlite3_column_text(statement,0)));
                }
            }
```

При выводе массива outTableCoffeeShop получим:

```
["SKURATOV", "LES", "ABC Coffee Roasters", "Starbucks", "Bloom-n-Brew"]
```

Закрытие БД

```
    if sqlite3_close(db) != SQLITE_OK {
            print("error closing database")
        }

        db = nil
```

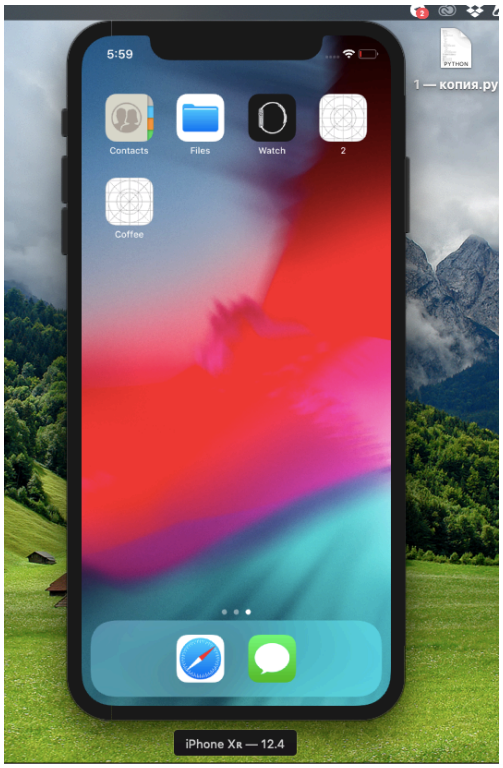
Возвращают SQLITE_OK, если объект sqlite3 успешно уничтожен и все связанные ресурсы освобождены.

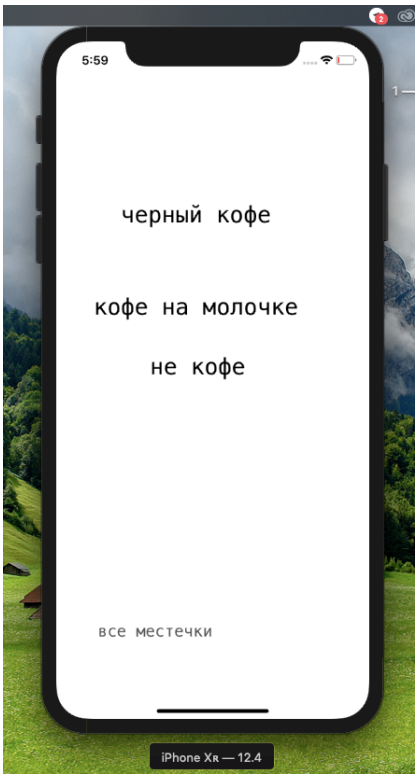
# Разработка приложения

Всю работу будем выполнять в приложении Xcode
При создании я стараюсь придерживаться минималистично стиля

Добавим компоненты и связи между ними:

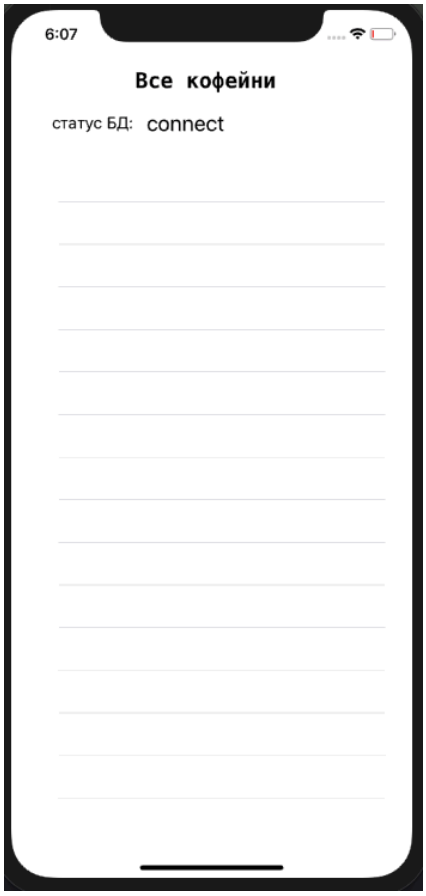

Проверять работоспособность будет с помощью симулятора телефона



В начале будем проверять подключение к базе данных и извлечение информации из нее в таблицу

Нажав на кнопку «все местечки» мы прийдем к ному окну: Все кофейни



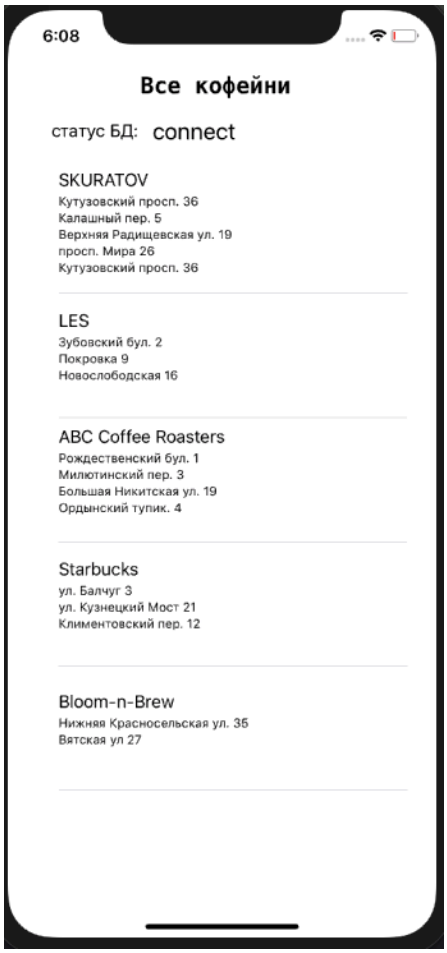

Пропишем в коде:

```
if sqlite3_open(fileURL.path, &db) == SQLITE_OK
{
        StatusDB text = "connect"
}
```

Как видно в Label (StatusDB) выводится текст, говорящий о том, что подключение прошло успешно

С помощью этого таких строк получим массив outTableCoffeeShop, который будет содержать названия всех сетей:

```
 if sqlite3_prepare_v2(db, "select name from
coffee_shop_chain;", -1, &statement, nil) == SQLITE_OK {
                while sqlite3_step(statement) == SQLITE_ROW
                {

outTableCoffeeShop.append(String(cString:
sqlite3_column_text(statement,0)));
                }
                print(outTableCoffeeShop)
```




А потом в цикле for, итерируясь по названиям сетей

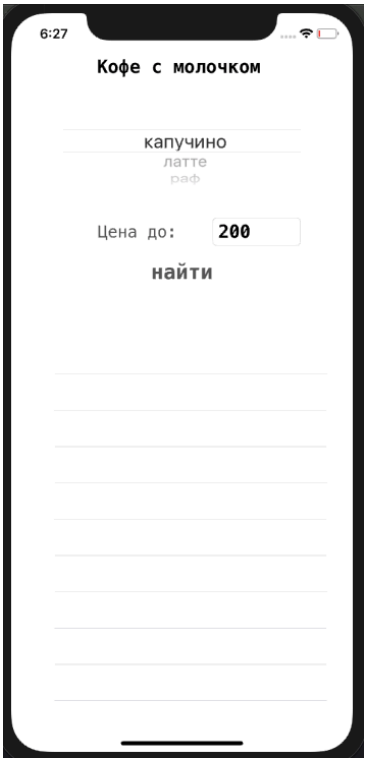
(["SKURATOV", "LES", "ABC Coffee Roasters", "Starbucks", «Bloom-n-Brew»])

Заполним матрицу со строками переменной длины ( из-за разного кол-во кофеен у сети) информацию про каждое заведение

```
var i = 0 //счетчик по сетям кофеен
            for coffee_shop in outTableCoffeeShop{

                select = "select street, house from cafe
join coffee_shop_chain on cafe.coffee_shop_chain_id =
coffee_shop_chain.id where name = '" + coffee_shop + "';"
                if sqlite3_prepare_v2(db, select , -1,
&statement, nil) == SQLITE_OK {

                outTable.append([]);
                while sqlite3_step(statement) == SQLITE_ROW
                {
                    outTable[i].append(String(cString:
sqlite3_column_text(statement,0)));//адрес

                }
                i = i + 1 }}
```

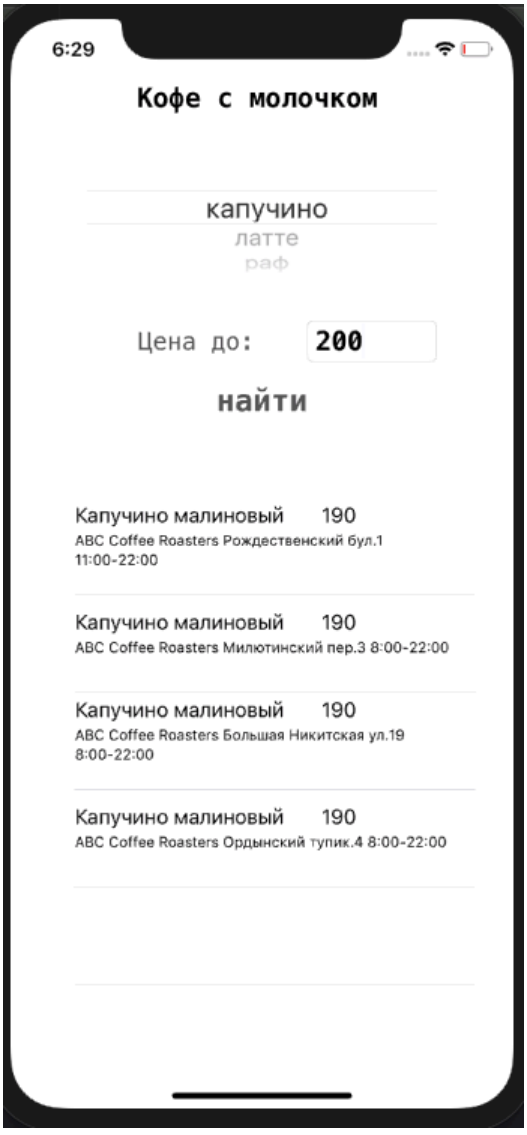
Перейдем на другую вкладку:

Задав напиток и ограничение в цене, получим список мест кофеен с адресом и временем работы.




Пример запросов:

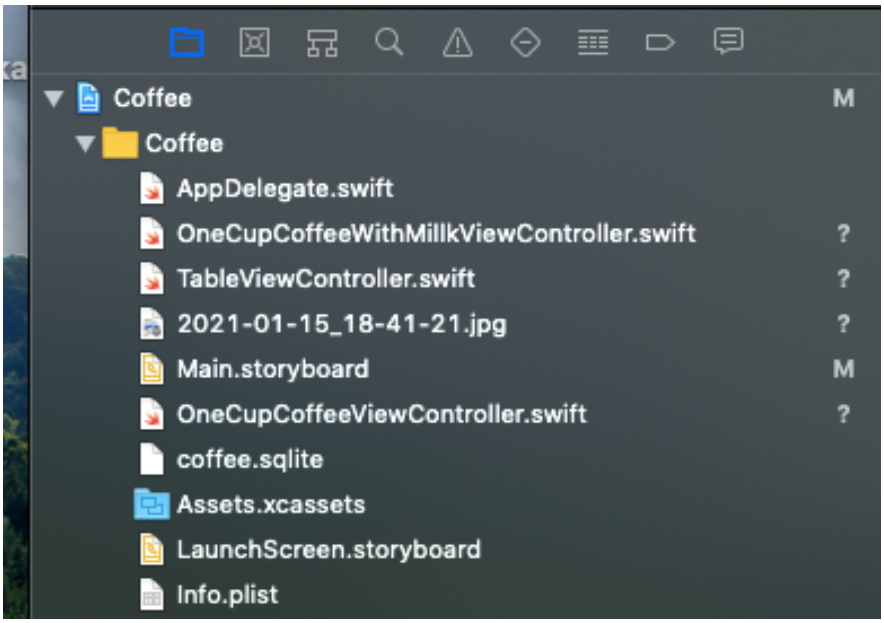
( если список большой, то можно перемотать вниз )




Финальный вид меню будет выглядеть так:

# Код приложения

Файлы содержащиеся в проекте:




```
import UIKit
import SQLite3

class OneCupCoffeeWithMillkViewController: UIViewController,UIPickerViewDelegate,
UIPickerViewDataSource {
    func numberOfComponents(in pickerView: UIPickerView) -> Int {
        return 1
    }

    func pickerView(_ pickerView: UIPickerView, numberOfRowsInComponent component: Int) -> Int {
        return MilkDrink.count
    }

    func pickerView(_ pickerView: UIPickerView, titleForRow row: Int, forComponent component: Int)
-> String? {
        return MilkDrink[row]
    }
    func pickerView(_ pickerView: UIPickerView, didSelectRow row: Int, inComponent component: Int) {
        return FixName = MilkDrink[row]
    }

    var MilkDrink :[String] = []
    //для таблицы вывода
    var HeaderDrink :[[String]] = []
    var Drink:[[String]] = []
    let idCell = "cell"
    var selectStr = ""// для запроса


    @IBOutlet weak var PickerCoffeeWithMilk: UIPickerView!
    @IBOutlet weak var FixPrice: UITextField!
    var FixName: String = ""
```

```
    @IBAction func SeachCoffee(_ sender: Any) {
        //работа с БД
        let fileURL = URL.init(fileURLWithPath: Bundle.main.path(forResource: "coffee", ofType:
"sqlite")!)
        // open database
        var db: OpaquePointer?

        var i = 0
        if sqlite3_open(fileURL.path, &db) == SQLITE_OK {
            var statement: OpaquePointer?
            // обрабатываем запрос напиток + цена
            selectStr = "select
drink.name,coffee_content.name,cafe_drink.price,coffee_shop_chain.name, cafe.street,cafe.house,
cafe.opening_time,cafe.closing_time from drink inner join cafe_drink ON drink_id = drink.id inner
join base on base.id = drink.base_id inner join cafe ON cafe_id = cafe.id inner join coffee_content
ON coffee_content_id = coffee_content.id inner join coffee_shop_chain ON coffee_shop_chain_id =
coffee_shop_chain.id  inner join name_drink  on name_drink.id = drink.name_drink_id  where
cafe_drink.price < " + (FixPrice.text!) + "  AND  name_drink.name = '" + FixName + "' AND base.id
NOT IN ( 1, 6) AND  coffee_content.id NOT IN (1);"
            if (Drink.count != 0){
                Drink = []
                HeaderDrink = []
            }
            if sqlite3_prepare_v2(db, selectStr, -1, &statement, nil) == SQLITE_OK
            {
                while sqlite3_step(statement) == SQLITE_ROW
                {
                    Drink.append([])
                    HeaderDrink.append([])
                    HeaderDrink[i].append(String(cString: sqlite3_column_text(statement,0))); //
drink.name
                    //HeaderDrink[i].append(String(cString: sqlite3_column_text(statement,1))); //
coffee_content.name
                    HeaderDrink[i].append(String(cString: sqlite3_column_text(statement,2))); //
cafe_drink.price
                    Drink[i].append(String(cString: sqlite3_column_text(statement,3))); //
coffee_shop_chain.name
                    Drink[i].append(String(cString: sqlite3_column_text(statement,4))); //
cafe.street
                    Drink[i].append(String(cString: sqlite3_column_text(statement,5))); //cafe.house
                    Drink[i].append(String(cString: sqlite3_column_text(statement,6))); //
cafe.opening_time
                    Drink[i].append(String(cString: sqlite3_column_text(statement,7))); //
cafe.closing_time
                    i = i+1
                }
            }
            print(HeaderDrink)
            print(Drink)
            print(HeaderDrink.count)

            if sqlite3_close(db) != SQLITE_OK {
                print("error closing database")
            }

            db = nil
        }
        Table.reloadData()

    }
    @IBOutlet weak var Table: UITableView!

    override func viewDidLoad() {
        super.viewDidLoad()
        Table.dataSource = self
        Table.delegate = self
        //работа с БД
        let fileURL = URL.init(fileURLWithPath: Bundle.main.path(forResource: "coffee", ofType:
"sqlite")!)
        // open database
        var db: OpaquePointer?

        if sqlite3_open(fileURL.path, &db) == SQLITE_OK {
            var statement: OpaquePointer?
            // заполняем список напитков
            if sqlite3_prepare_v2(db, "  select name_drink.name from drink inner join base on
base.id = drink.base_id  inner join name_drink  on name_drink.id = drink.name_drink_id inner join
coffee_content ON coffee_content_id = coffee_content.id where base.id NOT IN ( 1, 6) AND
coffee_content.id NOT IN (1) GROUP BY name_drink.name;", -1, &statement, nil) == SQLITE_OK
            {
```

```
                while sqlite3_step(statement) == SQLITE_ROW
                {
                    MilkDrink.append(String(cString: sqlite3_column_text(statement,0)));
                }
            }
            print(MilkDrink)
            if sqlite3_close(db) != SQLITE_OK {
                print("error closing database")
            }
            db = nil
        }
        FixName = MilkDrink[0]
        FixPrice.text = "200"
    }
}

extension OneCupCoffeeWithMillkViewController: UITableViewDataSource, UITableViewDelegate{
    func tableView(_ tableView: UITableView, numberOfRowsInSection section: Int) -> Int {
        return HeaderDrink.count
    }

    func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath) -> UITableViewCell {
        let cell = UITableViewCell(style: .subtitle, reuseIdentifier: idCell)

        var OneCoffee: String = ""
        for i in stride(from: 0, to: HeaderDrink[indexPath.row].count, by: 2){
            OneCoffee+=HeaderDrink[indexPath.row][i] // напиток
            //OneCoffee+=" "
            //OneCoffee+=HeaderDrink[indexPath.row][i+1] // зерна
            OneCoffee+="       "
            OneCoffee+=HeaderDrink[indexPath.row][i+1] // цена
            OneCoffee+="\n"
            print(OneCoffee)
        }
        cell.textLabel?.text = OneCoffee //заполняем название сети

        var ForOneCoffee: String = ""
        for i in stride(from: 0, to:Drink[indexPath.row].count, by: 5){
            ForOneCoffee+=Drink[indexPath.row][i] // сеть
            ForOneCoffee+=" "
            ForOneCoffee+=Drink[indexPath.row][i+1] // адресс
            ForOneCoffee+=Drink[indexPath.row][i+2] //  дом
            ForOneCoffee+=" "
            ForOneCoffee+=Drink[indexPath.row][i+3] // время открытия
            ForOneCoffee+="-"
            ForOneCoffee+=Drink[indexPath.row][i+4] // время закрытия
            ForOneCoffee+="\n"
            print(ForOneCoffee)
        }
        cell.detailTextLabel!.numberOfLines = Drink[indexPath.row].count
        cell.detailTextLabel!.text = ForOneCoffee
        return cell
}
    func tableView(_ tableView: UITableView, heightForRowAt indexPath: IndexPath) -> CGFloat {
        return 80.0
    }
}
```

```
import UIKit
import SQLite3

class TableViewController: UIViewController {

    var outTableCoffeeShop: [String] = []
    var outTable: [[String]] = []
    var select = ""
    @IBOutlet weak var StatusDB: UILabel!
    let idCell = "cell"

    @IBOutlet weak var Table: UITableView!
    override func viewDidLoad() {
        super.viewDidLoad()
        Table.dataSource = self
        Table.delegate = self

        //работа с БД
        let fileURL = URL.init(fileURLWithPath: Bundle.main.path(forResource: "coffee", ofType:
"sqlite")!)
        // open database
        var db: OpaquePointer?
        if sqlite3_open(fileURL.path, &db) == SQLITE_OK {
            StatusDB.text = "connect"
            var statement: OpaquePointer?
            //заполняем название сетей кофейн
            if sqlite3_prepare_v2(db, "select name from coffee_shop_chain;", -1, &statement, nil) ==
SQLITE_OK {
                while sqlite3_step(statement) == SQLITE_ROW
                {
                    outTableCoffeeShop.append(String(cString: sqlite3_column_text(statement,0)));
                }
                print(outTableCoffeeShop)
            }
            var i = 0 //счетчик по сетям кофеен
            for coffee_shop in outTableCoffeeShop{

                select = "select street,house from cafe join coffee_shop_chain on
cafe.coffee_shop_chain_id = coffee_shop_chain.id where name = '" + coffee_shop + "';"
                if sqlite3_prepare_v2(db, select , -1, &statement, nil) == SQLITE_OK {

                outTable.append([]);
                while sqlite3_step(statement) == SQLITE_ROW
                {
                    outTable[i].append(String(cString: sqlite3_column_text(statement,0)));//адрес

                }
                i = i + 1
                }}
            }
        if sqlite3_close(db) != SQLITE_OK {
            print("error closing database")
        }

        db = nil
        }
    }

extension TableViewController: UITableViewDataSource, UITableViewDelegate{
    func tableView(_ tableView: UITableView, numberOfRowsInSection section: Int) -> Int {
        return outTableCoffeeShop.count
    }
    func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath) -> UITableViewCell {
        let cell = UITableViewCell(style: .subtitle, reuseIdentifier: idCell)
        let one = outTableCoffeeShop[indexPath.row]
        cell.textLabel?.text = one //заполняем название сети

        var coffee_chain: String = ""
        for i in stride(from: 0, to:outTable[indexPath.row].count, by: 2){
            coffee_chain+=outTable[indexPath.row][i] // адрес
            coffee_chain+=" "
            coffee_chain+=outTable[indexPath.row][i+1] // дом
            coffee_chain+="\n"
            print(coffee_chain)
        }
        cell.detailTextLabel!.numberOfLines = outTable[indexPath.row].count
        cell.detailTextLabel!.text = coffee_chain
        return cell
    }
    func tableView(_ tableView: UITableView, heightForRowAt indexPath: IndexPath) -> CGFloat {
        return 120.0 }}
```

```
import UIKit
import SQLite3

class OneCupCoffeeViewController: UIViewController, UIPickerViewDelegate, UIPickerViewDataSource {

    @IBOutlet weak var PicName: UIPickerView!
    var NonMilkDrink :[String] = []
    //для таблицы вывода
    var HeaderDrink :[[String]] = []
    var Drink :[[String]] = []
    //var WithMilkDrink:[String] = []
    let idCell = "cell"
    @IBOutlet weak var Table: UITableView!
    @IBOutlet weak var ForPrice: UISlider!
    @IBOutlet weak var FixName: UITextField! // имя напитка
    @IBOutlet weak var FixPrice: UITextField! // предел цены

    var selectStr = ""// для запроса

    func numberOfComponents(in pickerView: UIPickerView) -> Int {
        return 1
    }

    func pickerView(_ pickerView: UIPickerView, numberOfRowsInComponent component: Int) -> Int {
        return NonMilkDrink.count
    }
    func pickerView(_ pickerView: UIPickerView, titleForRow row: Int, forComponent component: Int)
-> String? {
        return NonMilkDrink[row]
    }

    func pickerView(_ pickerView: UIPickerView, didSelectRow row: Int, inComponent component: Int) {
        return FixName.text = NonMilkDrink[row]
    }

    @IBAction func ButtSeach(_ sender: Any) {
        Table.dataSource = self
        Table.delegate = self
        //работа с БД
        let fileURL = URL.init(fileURLWithPath: Bundle.main.path(forResource: "coffee", ofType:
"sqlite")!)
        // open database
        var db: OpaquePointer?

        var i = 0
        if sqlite3_open(fileURL.path, &db) == SQLITE_OK {
            var statement: OpaquePointer?
            // обрабатываем запрос напиток + цена
            print(FixPrice.text!)
            selectStr = "select
drink.name,coffee_content.name,cafe_drink.price,coffee_shop_chain.name, cafe.street,cafe.house,
cafe.opening_time,cafe.closing_time from drink inner join cafe_drink ON drink_id = drink.id inner
join base on base.id = drink.base_id inner join coffee_content ON coffee_content_id =
coffee_content.id inner join cafe ON cafe_id = cafe.id inner join coffee_shop_chain ON
coffee_shop_chain_id = coffee_shop_chain.id  where cafe_drink.price < " + (FixPrice.text!) + " AND
drink.name = '" + (FixName.text!) + "' AND  base.id IN ( 1, 6);"
            if (Drink.count != 0){
                Drink = []
                HeaderDrink = []
            }
            if sqlite3_prepare_v2(db, selectStr, -1, &statement, nil) == SQLITE_OK
            {
                while sqlite3_step(statement) == SQLITE_ROW
                {
                    Drink.append([])
                    HeaderDrink.append([])
                    HeaderDrink[i].append(String(cString: sqlite3_column_text(statement,0))); //
drink.name
                    HeaderDrink[i].append(String(cString: sqlite3_column_text(statement,1))); //
coffee_content.name
                    HeaderDrink[i].append(String(cString: sqlite3_column_text(statement,2))); //
cafe_drink.price
                    Drink[i].append(String(cString: sqlite3_column_text(statement,3))); //
coffee_shop_chain.name
                    Drink[i].append(String(cString: sqlite3_column_text(statement,4))); //
cafe.street
                    Drink[i].append(String(cString: sqlite3_column_text(statement,5))); //cafe.house
                    Drink[i].append(String(cString: sqlite3_column_text(statement,6))); //
cafe.opening_time
                    Drink[i].append(String(cString: sqlite3_column_text(statement,7))); //
cafe.closing_time
```

```
                    i = i+1
                }
            }
            if sqlite3_close(db) != SQLITE_OK {
                print("error closing database")
            }

            db = nil
        }
        Table.reloadData()
}
    override func viewDidLoad() {
        super.viewDidLoad()
        Table.dataSource = self
        Table.delegate = self
        //работа с БД
        let fileURL = URL.init(fileURLWithPath: Bundle.main.path(forResource: "coffee", ofType:
"sqlite")!)
        // open database
        var db: OpaquePointer?

        if sqlite3_open(fileURL.path, &db) == SQLITE_OK {
            var statement: OpaquePointer?
            // заполняем список напитков
                if sqlite3_prepare_v2(db, "select drink.name from drink join base on base.id =
drink.base_id where base.id IN ( 1, 6) GROUP BY drink.name;", -1, &statement, nil) == SQLITE_OK
                {
                    while sqlite3_step(statement) == SQLITE_ROW
                    {
                        NonMilkDrink.append(String(cString: sqlite3_column_text(statement,0)));
                    }
                 }
            if sqlite3_close(db) != SQLITE_OK {
                print("error closing database")
            }
            db = nil
          }
        FixName.text = NonMilkDrink[0]
        FixPrice.text = "200"
}}
extension OneCupCoffeeViewController: UITableViewDataSource, UITableViewDelegate{
    func tableView(_ tableView: UITableView, numberOfRowsInSection section: Int) -> Int {
         return HeaderDrink.count
    }

    func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath) -> UITableViewCell {
        let cell = UITableViewCell(style: .subtitle, reuseIdentifier: idCell)

        var OneCoffee: String = ""
        for i in stride(from: 0, to: HeaderDrink[indexPath.row].count, by: 3){
            OneCoffee+=HeaderDrink[indexPath.row][i] // напиток
            OneCoffee+=" "
            OneCoffee+=HeaderDrink[indexPath.row][i+1] // зерна
            OneCoffee+="       "
            OneCoffee+=HeaderDrink[indexPath.row][i+2] // цена
            OneCoffee+="\n"
        }
        cell.textLabel?.text = OneCoffee //заполняем название сети

        var ForOneCoffee: String = ""
        for i in stride(from: 0, to:Drink[indexPath.row].count, by: 5){
            ForOneCoffee+=Drink[indexPath.row][i] // сеть
            ForOneCoffee+=" "
            ForOneCoffee+=Drink[indexPath.row][i+1] // адресс
            ForOneCoffee+=Drink[indexPath.row][i+2] //  дом
            ForOneCoffee+=" "
            ForOneCoffee+=Drink[indexPath.row][i+3] // время открытия
            ForOneCoffee+="-"
            ForOneCoffee+=Drink[indexPath.row][i+4] // время закрытия
            ForOneCoffee+="\n"
            print(ForOneCoffee)
        }
        cell.detailTextLabel!.numberOfLines = Drink[indexPath.row].count
        cell.detailTextLabel!.text = ForOneCoffee
        return cell
    }

        func tableView(_ tableView: UITableView, heightForRowAt indexPath: IndexPath) -> CGFloat {
            return 80.0
        }
}
```

# Содержание


Постановка задачи 2

Базы данных 3

Проектирование базы данных 3

Создание и заполнение БД 8

Запросы SQL 9

Программный доступ 10

Разработка приложения 13

Код приложения 17

Содержание 23