

Лекция №4. Введение в SQL

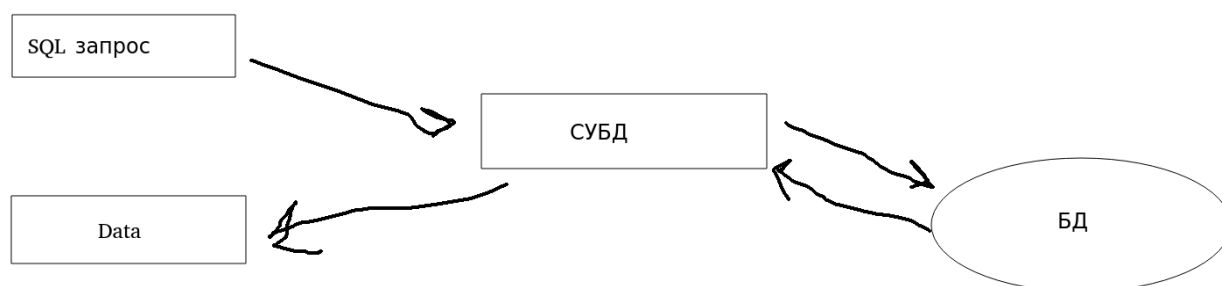
Автор: Набиев Марат

Базы данных(БД) — набор сведений, хранящихся некоторым упорядоченным образом. Можно сравнить БД со шкафом, в котором хранятся документы, т. е. БД - хранилище данных. БД сами по себе неинтересны, т. к. с просто хранящимися данными ничего полезного не сделаешь.

Система управления базами данных (СУБД) — совокупность языковых и программных средств, которые осуществляют доступ к данным, позволяет их добавлять, менять, удалять, обеспечивает безопасность. Таким образом, **СУБД** — это система, позволяющая создавать БД и манипулировать сведениями из них. Доступ к данным осуществляется посредством языка SQL.

SQL — язык структурных запросов, основной задачей которого является предоставление способа считывания и записи информации базу данных.

Простейшая схема как работают запросы:



Наиболее популярные СУБД: MySQL, PostgreSQL, MS SQL, Oracle, MongoDB, SQLite

Создавая БД, мы стремимся упорядочить информацию по различным признакам, чтобы потом извлекать нужные нам данные в любом сочетании. Для этого надо хорошо структурировать данные. Существуют много способов это сделать: иерархическая, объектная, объектно-ориентированная, реляционная, сетевая, функциональная и д.р.

Мы рассмотрим только реляционные базы данных.

Реляционные БД

В реляционных БД данные представлены в виде простых таблиц, разбитых на строки и столбцы, на пересечении которых лежат данные. Столбцы называются **полями или атрибутами**, строки — **записи или кортежи**.

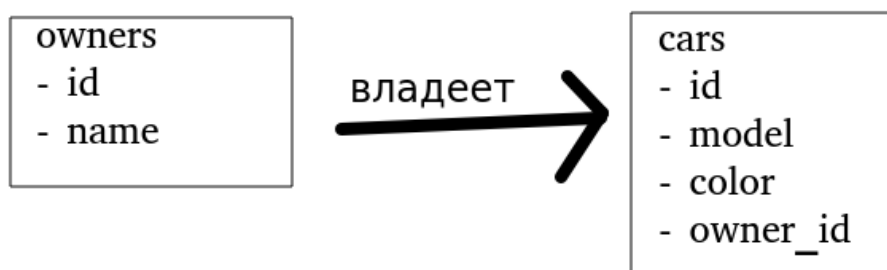
Таблицы обладают следующими свойствами:

- нет 2 одинаковых строк
- столбцы расположены в определенном порядке, который задается при создании таблицы
- может быть ни одной строки, но хотя бы 1 столбец должен быть
- у каждого столбца есть уникальное имя в пределах таблицы и имеет определенный тип.
- в каждой ячейке только атомарное (неделимое) значение

В базе данных могут быть десятки и сотни таблиц. Для удобства в начале проектируют базу данных (определяют отношения между таблицами) и потом только переходят к непосредственной реализации.

Далее нарисует «архитектуру» нашей БД, перейдем к запросам, работать будем с СУБД SQLite, которая не требует поднятия серверов и прочего, а просто хранится на пользовательском компьютере в виде файла с расширением .db, .sqlite, sqlite3 и т. п.

Наша БД.



Далее что мы будем делать:

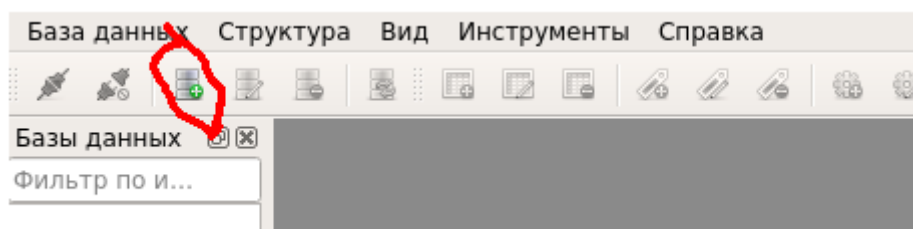
- 1) создадим БД
- 2) добавим таблицы owners и cars
- 3) заполним данными
- 4) удалим что-то и изменим
- 5) попробуем получить различные комбинации данных

Будем работать через SQLiteStudio (который написан на Qt) ссылка на сайт:

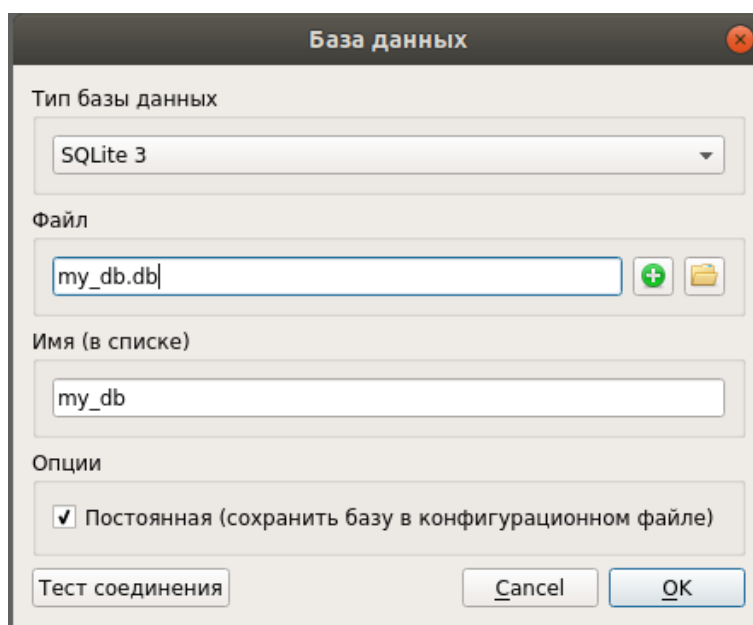
<https://sqlitestudio.pl/>

Создание БД.

Запускаем SQLiteStudio, и добавляем БД, нажав на значок базы данных с плюсиком



Пишем название и сохраняем

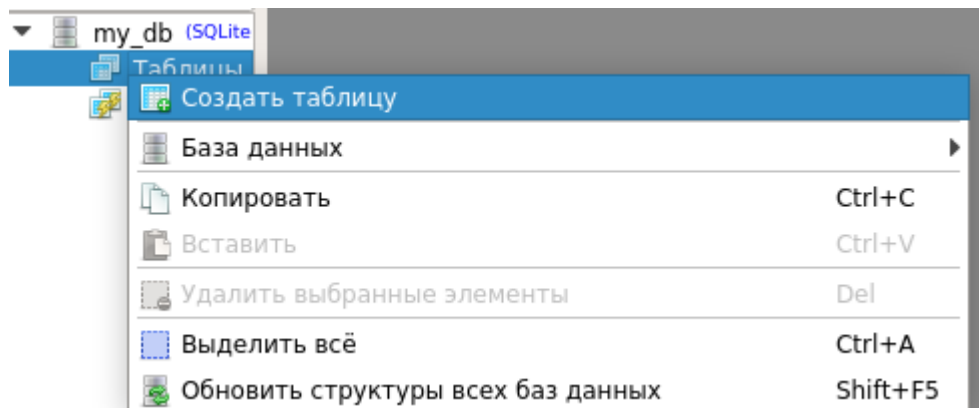


Подключаемся к БД, нажав на значок соединения.

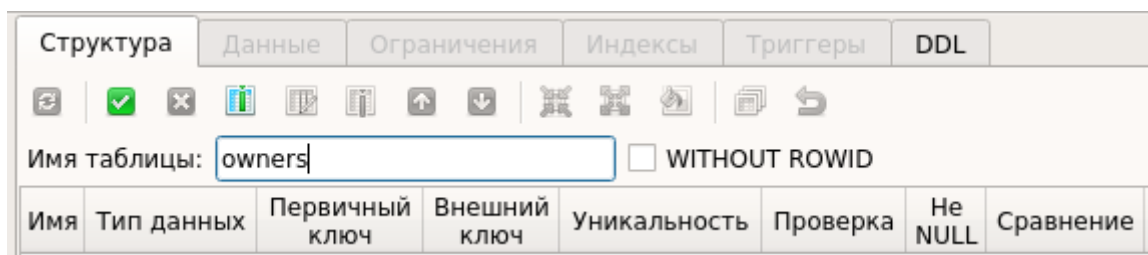
(там типа 2 провода, лень скрин делать)

Создание таблиц

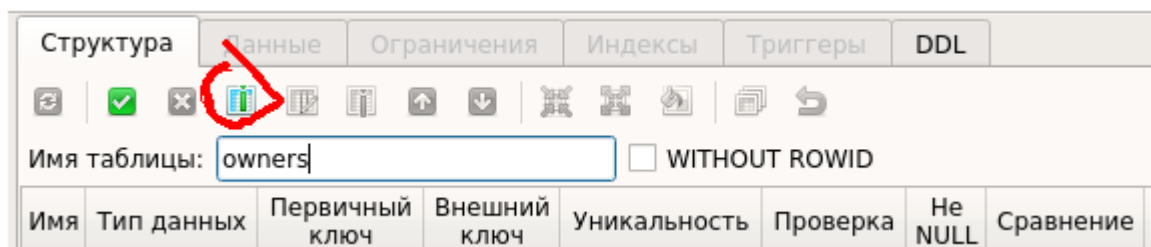
Теперь нажимаем на таблицы правой кнопкой мыши и выбираем создать таблицу



Напишем название таблицы



Теперь надо добавить столбцы. Для этого нажимаем на эту кнопку



Пишем название столбца `id` и тип `INTEGER`.

Теперь поставим галочку на «Первичный ключ» и нажмем на кнопку настроить, где ставим галочка на «Автоинкремент»

Столбец

Имя и тип

Имя столбца: Тип данных: INTEGER Размер: ,

Ограничения

- ☒ Первичный ключ Настроить
- ☐ Внешний ключ Настроить
- ☐ Уникальность Настроить
- ☐ Проверка условия Настроить
- ☐ Не NULL Настроить
- ☐ Сравнение Настроить
- ☐ Default Настроить

☐ Расширенный режим Cancel OK

Редактировать ограничение

Первичный ключ

☒ Автоинкремент

☐ Порядок сортировки: ASC













☐ Именованное ограничение:

☐ При конфликте: ROLLBACK

Cancel Применить

Default Настроить

Нажимаем на Ок и видим, что у нас есть один столбец.













Структура								
<div>Данные</div> <div>Ограничения</div> <div>Индексы</div> <div>Триггеры</div> <div>DDL</div>								
<div>          </div>								
Имя таблицы: <input type="text" value="owners"/> <input type="checkbox"/> WITHOUT ROWID								
	Имя	Тип данных	Первичный ключ	Внешний ключ	Уникальность	Проверка	Не NULL	Сравнение
1	id	INTEGER						NULL

Что такое первичный ключ?

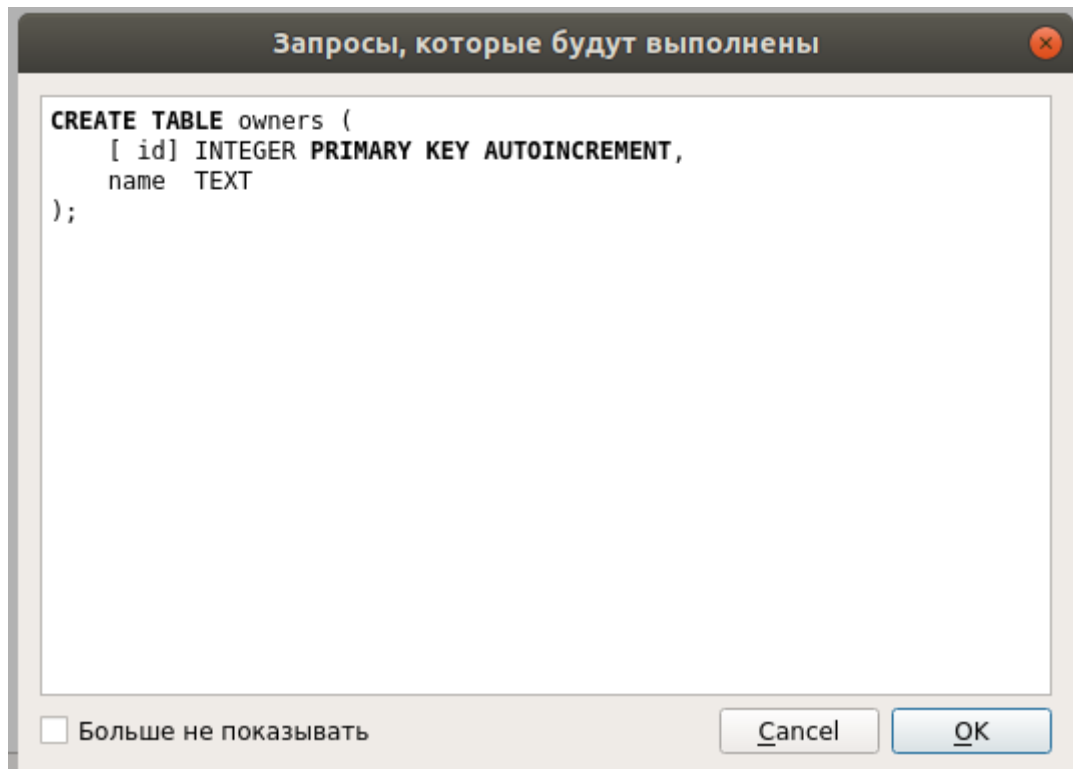
Первичный ключ (primary key) - это уникальный идентификатор записи в таблице. Первичным ключом могут быть сразу группа столбцов, но обычно за это отвечает столбец `id`.

Автоинкремент добавляется для этого, чтобы мы сами не придумывали `id`, а он увеличивался автоматически.

Теперь добавим поле `name`, сделаем тип `TEXT`, и это поле, конечно, не является ключом, т. к. может быть много людей с одинаковыми именами.

Структура								
<div>Данные</div> <div>Ограничения</div> <div>Индексы</div> <div>Триггеры</div> <div>DDL</div>								
<div>          </div>								
Имя таблицы: <input type="text" value="owners"/> <input type="checkbox"/> WITHOUT ROWID								
	Имя	Тип данных	Первичный ключ	Внешний ключ	Уникальность	Проверка	Не NULL	Сравнение
1	id	INTEGER						NULL
2	name	TEXT						NULL

Получаем такую таблицу, нажимаем на зеленый квадратик с галочкой.



И SQLiteStudio генерирует нам запрос для создания таблицы, этот запрос мы можем написать и сами, но мы пошли другим путем.

Теперь создадим таблицу `cars`, в котором будут столбцы `id` (тип `INTEGER`), `model` (`TEXT`), `color` (`TEXT`), `owner_id` (`INTEGER`). Думаю, вы справитесь сами.

Получим такую табличку

Структура									
Данные									
Ограничения									
Индексы									
Триггеры									
DDL									
Имя таблицы: cars <input type="checkbox"/> WITHOUT ROWID									
Имя	Тип данных	Первичный ключ	Внешний ключ	Уникальность	Проверка	Не NULL	Сравнение	Значение по умолчанию	
1 id	INTEGER							NULL	
2 model	TEXT							NULL	
3 color	TEXT							NULL	
4 owner_id	INTEGER							NULL	

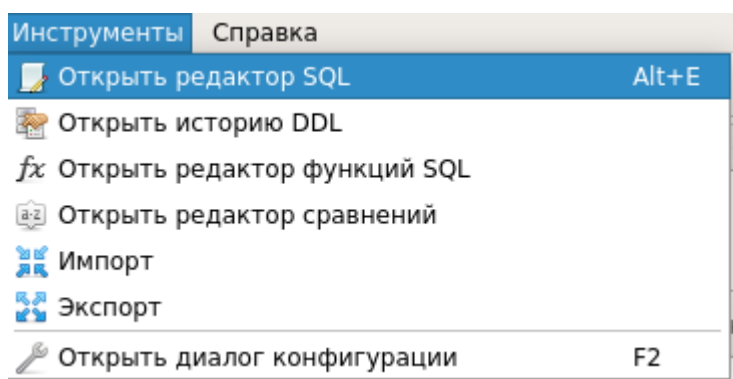
Заполнение таблиц.

Пустые таблицы никому не интересны. Нам нужны данные, чтобы было с чем работать. Для этого существует команда `INSERT`. У нее следующий синтаксис (обычно для лучшей читабельности команды `sql` пишут капсом, а названия таблиц, полей строчными)

```
INSERT INTO table_name (перечислить поля)
VALUES (данные в том порядке, какие в были выше) .
```

Поля перечислять необязательно, если вы вводите данные в том же порядке, в каком расположены столбцы.

Для того, чтобы вводить команды `sql`, в меню-баре нажимаем на «Инструменты» и там выбираем «Открыть редактор SQL»



Открывается такое окошко. Сверху пишут запросы, а внизу результат.

Напишем добавление нового человека в таблицу `owners`.

```
1 INSERT INTO owners (name)
2 VALUES ('Victor');
```

Такая команда, строки пишутся в одинарных кавычках, и `id` не указываем, т. к. он генерируется автоматически. Для выполнения надо нажать на синий треугольник.

И добавим еще одного владельца. Назовем его Peter.

Запрос **SELECT**

Самый популярный запрос. Синтаксис примерно следующий

SELECT поля

FROM table

WHERE какие-то условия, необязательно (забыл сказать про него на паре)

ORDER BY (для сортировки, необязательно)

GROUP BY (для группирования, тоже не обязательно)

Получим все поля из таблицы `owners`, для этого пишем следующий запрос:

(* означает все поля)

```
1 SELECT *
2 FROM owners;
```

Результат будет следующий:

	id	name
1	1	Victor
2	2	Peter

Теперь сами добавьте данные в таблицу cars (в owner_id пишете 1, если вы хотите присвоить машину владельцу с id =1)

При помощи `SELECT * FROM cars` выведем что у нас получилось. Я добавил следующие тачки

	id	model	color	owner_id
1	1	VAZ-2106	blue	1
2	2	VAZ-2105	red	1
3	3	ZAZ-968	orange	2
4	4	BMW x6	orange	2
5	5	GAZ-53	orange	2
6	6	Moskvich-412	black	1

Команда DELETE

Допустим, владелец 1 продал свой москвич и надо удалить эту запись из таблицы, для этого есть команда `DELETE`. Синтаксис примерно следующий

`DELETE FROM table_name`

`WHERE условие`

Удалим машину с `id = 6`. (для сравнения пишем только 1 равно), можно необязательно писать такое условие. Условия могут быть различны, они ограничены лишь вашей фантазией

Такой запрос:

```
1 DELETE FROM cars
2 WHERE id = 6;
```

И если наберете `SELECT * FROM cars`, то увидите, что этой записи уже нет.

Команда UPDATE

Также данные в таблице могут меняться, допустим, владелец 2 решил, что оранжевый BMW — это несерьезно и перекрасил в зеленый. Для таких случаев есть команда `UPDATE`.

Синтаксис:

UPDATE table

SET пишем, что меняем

WHERE условие.

Мы напишем следующий запрос:

```
1 UPDATE cars
2 SET color = 'green'
3 WHERE model LIKE 'BMW%';
```

в итоге в нашей таблице все машины, модель которой начинается с BMW перекрасятся в зеленый.

Конструкция LIKE „...“ для выбора строк, «BMW%» - значит что слово начинается с BMW и неважно что там дальше.

В результате наши данные будут следующими:

	id	model	color	owner_id
1	1	VAZ-2106	blue	1
2	2	VAZ-2105	red	1
3	3	ZAZ-968	orange	2
4	4	BMW x6	green	2
5	5	GAZ-53	orange	2

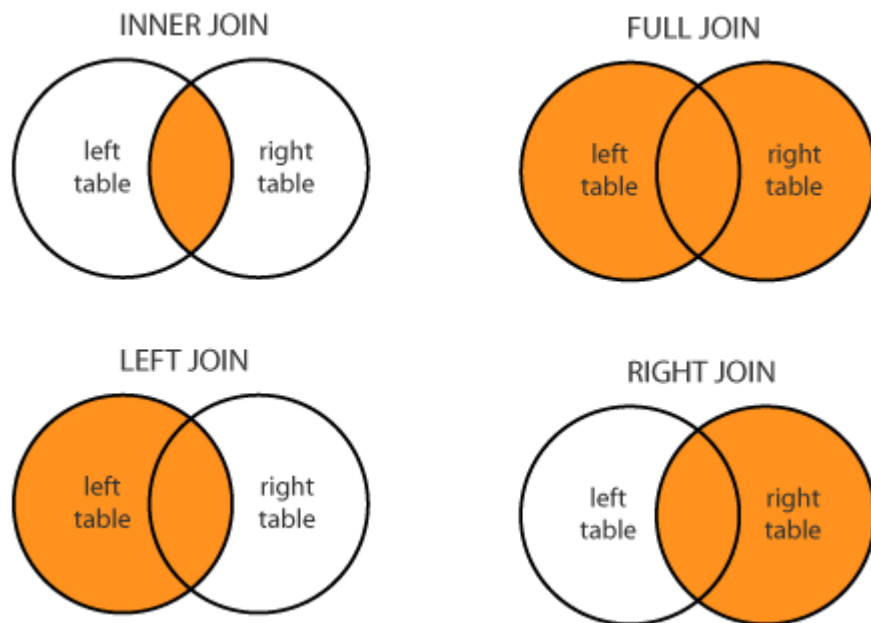
Вернемся к **SELECT**.

При помощи этой команды можно получать самые разные, интересные комбинации данных.

JOIN

Когда мы выводим данные таблицы `cars`, `owner_id` нам мало о чем говорит, мы бы хотели видеть имена владельцев машин. Для этого используют связывание таблиц. Бывают `LEFT`, `RIGHT`, `INNER` и даже говорят `FULL join`-ы

Следующая картинка описывает их



мы напишем `INNER JOIN` (наверно наиболее часто употребляемый)

Синтаксис:

`SELECT` поля

`FROM t1 INNER (или любой другой вид) JOIN t2`

`ON` условия связывания

и т. д.

Запрос:

```
1 SELECT cars.id, model, color, owners.name
2 FROM cars INNER JOIN owners
3 ON cars.owner_id = owners.id;
4
```

Здесь встречается конструкция `table.field`, это для того, чтобы `id` из `cars` и `id` из `owners` разделялись.

Результат:

	id	model	color	name
1	1	VAZ-2106	blue	Victor
2	2	VAZ-2105	red	Victor
3	3	ZAZ-968	orange	Peter
4	4	BMW x6	green	Peter
5	5	GAZ-53	orange	Peter

Теперь посчитаем сколько машин у каждого владельца. Для этого есть функция `COUNT` и нам надо сгруппировать данные по `owner_id`. Будет это выглядеть следующим образом

```
1 SELECT COUNT(cars.id) as count, owners.name
2 FROM cars INNER JOIN owners
3 ON cars.owner_id = owners.id
4 GROUP BY cars.owner_id;
```

Что здесь произошло

- 1) мы находим количество машин `COUNT(cars.id) as count` (назвали этот столбец `count`)
- 2) чтобы не посчитать просто количество всех машин, группируем их по `owner_id`

Вложенные запросы

по сути, результат `SELECT` — это таблица, и поэтому мы можем написать вложенные запросы (`SELECT` внутри `SELECT`)

Допустим, мы хотим найти человека, у которого больше всего машин. У нас есть запрос который находит количество машин каждого, а теперь

просто обернем этот запрос в другой, который вернет нам человека, у которого больше всего машин.

Получается следующий запрос:

```
1 SELECT MAX(count) as max, name
2 FROM(
3     SELECT COUNT(cars.id) as count, owners.name
4     FROM cars INNER JOIN owners
5     ON cars.owner_id = owners.id
6     GROUP BY cars.owner_id
7 );
```

И результат будет следующим:

	max	name
1	3	Peter

Вроде все с введением. Конечно же здесь было описано далеко не все, это небольшое введение только. Исходников нет, т. к. все, что можно написать, здесь описал. Если что пишете.