**4.3. Работа с СУБД SQLite**

## Соединение с базой данных.

Подсоединиться к базе данных очень просто:  
  
db = openDatabase("ToDo", "0.1", "A list of to do items.", 200000);  
  
Данный код создаёт объект, представляющий БД, а если базы данных с таким именем не существует, то создаётся и она. При этом в аргументах указывается имя базы данных, версия, отображаемое имя и приблизительный размер. Кроме того важно отметить, что приблизительный размер не является ограничением. Реальный размер базы данных может изменяться.  
Успешность подключения к БД можно оценить, проверив объект db на null:  
  
if(!db){alert("Failed to connect to database.");}  
  
Всегда предпринимайте данную проверку, даже если соединение с БД для данного пользователя уже производилось в прошлом, и было успешно. Могут измениться настройки безопасности, закончиться дисковое пространство (скажем, если пользователь использует смартфон) или фаза луны окажется неподходящей.

## Выполнение запросов.

Для выполнения запросов к БД предварительно надо создать транзакцию, вызвав функцию database.transaction(). У неё один аргумент, а именно другая JavaScript функция, принимающая объект транзакции и предпринимающая запросы к базе данных.  
Собственно сам SQL запрос можно выполнить, вызвав функцию executeSql объекта транзакции. Она принимает 4 аргумента:

* строка SQL запроса
* массив параметров запроса (параметры подставляются на место вопросительных знаков в SQL запросе)
* функция, вызываемая при успешном выполнении запроса
* функция, вызываемая в случае возникновения ошибки выполнения запроса

***Пример работы функции executeSql ниже***:  
  
db.transaction(function(tx) {  
tx.executeSql("SELECT COUNT(\*) FROM ToDo", [], function(result){}, function(tx, error){});  
});  
  
Давайте теперь изменим код так, чтобы при невозможности выборки из таблицы «ToDo»(которой пока не существует), данная таблица создавалась.  
  
db.transaction(function(tx) {  
tx.executeSql("SELECT COUNT(\*) FROM ToDo", [], function (result) { alert('dsfsdf') }, function (tx, error) {  
tx.executeSql("CREATE TABLE ToDo (id REAL UNIQUE, label TEXT, timestamp REAL)", [], null, null);  
})});

## Вставка данных.

Давайте вставим новую строку в таблицу «ToDo». Для знакомых с синтаксисом SQL пример, приведённый ниже, покажется очень знакомым:  
  
db.transaction(function(tx) {  
tx.executeSql("INSERT INTO ToDo (label, timestamp) values(?, ?)", ["Купить iPad или HP Slate", new Date().getTime()], null, null);  
});  
  
Первый знак вопроса в SQL запросе заменяется на «Купить iPad или HP Slate», а второй на метку времени. В итоге выполнен будет примерно такой запрос:  
INSERT INTO ToDo (label, timestamp) values ("Купить iPad или HP Slate", 1265925077487)

## Работа с результатами запросов.

Результат выполнения запроса на выборку данных содержит набор строк, а каждая строка содержит значения столбцов таблицы для данной строки.  
Вы можете получить доступ к какой-либо строке результата вызвав функцию result.rows.item(i), где i – индекс строки. Далее, для получения требуемого значения, нужно обратиться к конкретному столбцу по имени – result.rows.item(i)[ «label»].  
Следующий пример выводит результат запроса к базе данных на страницу:  
db.transaction(function(tx) {  
tx.executeSql("SELECT \* FROM ToDo", [], function(tx, result) {  
for(var i = 0; i < result.rows.length; i++) {  
document.write('<b>' + result.rows.item(i)['label'] + '</b><br />');  
}}, null)});  
**WEB SQL в JavaScript**

0. Создать папку WEBDB

1. Создать файл index.html

2. Создать папку css

3. Создать папку js

4. В папке css содать файл style.css

5. В папке js создать файлы main.js, database.js, app.js

6. Наберем теги в файле index.html

<!DOCTYPE html>

<html lang=”en”>

<head>

<meta charset=”UTF-8”>

<meta name=”viewport” content=”width=device-width, initial-scale=1.0”>

<link rel=”stylesheet” href=”./css/style.css”>

<script src=”./js/database.js”></script>

<script src=”./js/app.js”></script>

<script src=”./js/main.js”></script>

<title>Document</title>

</head>

<body>

<div class=”app”>

<div class=”add-block”>

<div><input type=”text” id=”add-name” placeholder=”имя”></div>

<div><input type=”text” id=”add-surname” placeholder=”фамилия”></div>

<div><input type=”text” id=”add-mail” placeholder=”e-mail”></div>

<div><input type=”text” id=”add-phone-number” placeholder=”телефон”></div>

<div><input type=”button” id=”add” value=”+”></div><div></div>

</div>

<div class=”caption-table”>

<div>Имя</div><div>Фамилия</div>

<div>Mail/div><div>Телефон</div>

</div>

<div class=”table-block”><div class=”table”></div></div>

<div class=”clear-btn-block”><button id=”btnClear”>очистить</button></div>

</div>

</body>

7. В файле style.css опишем необходимые стили

html, body{

padding: 0;

margin: 0;

}

body{

background-color:#121212;

}

div.app{

width:100%;

background-color:#121212;

}

div.add-block{

display:grid;

grid-template-columns:1fr 1fr 1fr 1fr 24px;

}

div.caption-table{

display:grid;

grid-template-columns:1fr 1fr 1fr 1fr 54px;

color:#ccc;

background: linear-gradient(#121212, #333333);

font-weigtht: bold;

}

div.table{

display:grid;

grid-template-columns:1fr 1fr 1fr 1fr 54px;

color:#ccc;

}

div.table > div{

padding: 10px;

line-height:53px;

}

div.caption-table > div{

padding:10px;

}

div.clear-btn-block > button,

div.table > div > button,

div.add-block > div > input{

font-size:14px;

font-weight:bold;

padding: 8px;

width: 100%;

color:#ccc;

border: 0;

outline: none;

background-color: #121212;

border-bottom: 1px solid #ccc;

transition: all .5s;

}

div.add-block > div > input[type=”text”]:focus{

border-bottom: 1px solid #3a3aff;

transition: all .5s;

}

div.clear-btn-block > button:hover,

div.table > div > button:hover,

div.add-block > div > input[type=”button”]:hover{

border-bottom: 1px solid #3a3aff;

background-color:#333;

transition: all .5s;

}

div.clear-btn-block > button:active,

div.table > div > button:active,

div.add-block > div > input[type=”button”]:ative{

border-bottom: 1px solid #0000ff;

background-color:#000;

transition: all .5s;

}

div.table-block{

height:400px;

overflow-y: auto;

overflow-x: none;

}

8. Создать объект DataBase в файле database.js (конструктор объекта)

function DataBase(name){

this.dbName = name;

this.ucDbName = this.dbName=this.dbName[0].toUpperCase() + this.dbName.substring(1);

this.db = openDatebase(this.dbName, ‘1.0’, this.ucDbName, 100\*1024);

}

Конструктор объекта принимает имя базы данных, которое затем присваивается свойству ***dbName***. Свойству ***ucDbName*** присваивается тоже имя, но первая буква делается заглавной. Свойству ***db***присваивается дескриптор открытой базы. Функция ***openDatabase***открывает базу данных и принимает четыре параметра:

* имя базы данных;
* версия базы;
* описание;
* размер базы.

9. Создать метод create

DataBase.prototype.create = function(name){

var strFields = ‘ ‘;

var fields = {

name: ‘TEXT’, surname: ‘TEXT’,

email: ‘TEXT’, phone: ‘TEXT’,

}

for(const key in fields){

strFields += “,” + key + ‘ ‘ + fields[key];

}

this.db.transaction(function(tx){

tx.executeSql(‘create table if not exists’ + name + ‘(id integer primary key autoincrement’ + strFields + ‘)’);

});

};

Метод принимает имя создаваемой таблицы в виде параметра. В переменной ***strFields***будет содержаться ***sql***запрос. В объект ***fields*** поместить имена полей и тип хранимых данных. В данном случае у всех полей тип будет ***TEXT***. В цикле ***for***формируется строка запроса. Метод ***transaction***отправляет запрос. В ***sql***запросе присутствует условие ***create table if not exists*** - это значит, что таблица будет создана только в том случае. если она не существует. Создается метод ***add()***. В качестве параметра методу передается имя таблицы.

DataBase.prototype.add = function(name){

var q = new Array();

var vars = new Array();

var vals = new Array();

var values = {

name: document.getElementById(‘add-name’).value,

surname: document.getElementById(‘add-surname’).value,

email: document.getElementById(‘add-mail’).value,

phone: document.getElementById(‘add-phone-number’).value

}

for(var i = 0; I < Object.keys(values).length; i++){

q.push(‘?’);

}

for(const key in values){

vars.push(key);

}

for(const key in values){

vals.push(values[key]);

}

this.db.transaction(function(tx){

tx.executeSql(‘INSERT INTO ‘ + name + ‘(‘ + vars.join(“,”) + ‘) values(‘ + q.join(“,”) + ‘)’, vals);

});

};

Создаются три массива: ***q***, ***vars***и ***vals***. В методе values содержатся имена полей таблицы с присвоенными значениями, которые в свою очередь берутся из текстовых полей. В цикле ***for***заполняется массив ***q***знаками вопроса, массив ***vars***заполняется именами полей таблицы, а массив ***vals*** формируется значениями полей ввода. Далее отправляется ***sql***запрос. Создается метод ***load()***. Этот метод загружает данные из базы и выводит в виде таблицы.

Database.prototype.load = function(name){

var sql = ‘SELECT \* FROM $(name)’;

document.querySelector(‘.table).innerHTML = ‘ ‘;

this.db.transaction(function(tx){

tx.executeSql(sql, [ ], function(tx, result){

var n = result.rows.length;

for(var I = 0; I < n; i++){

var work = result.rows.item(i);

var tbl\_block = document.querySelector(‘.table’);

var cell\_name = document.createElement(‘div’);

var cell\_dell\_btn = document. createElement(‘div’);

var del\_btn = document. createElement(‘button’);

cell\_name.setAttribute(‘class’, ‘cell-tbaable’);

cell\_name.setAttribute(‘data-id’, work.id);

var cell\_surname = cell\_name.cloneNode(true);

var cell\_mail = cell\_name.cloneNode(true);

var cell\_phone = cell\_name.cloneNode(true);

cell\_name.setAttribute(‘data-field’, ‘name’);

cell\_surname.setAttribute(‘data-field’, ‘surname’);

cell\_mail.setAttribute(‘data-field’, ‘email’);

cell\_phone.setAttribute(‘data-field’, ‘phone’);

cell\_del\_btn.setAttribute(‘class’, ‘tbaable’);

del\_btn.setAttribute(‘data-id’, work.id);

cell\_name.InnerText = work.name;

cell\_surname.InnerText = work.surname;

cell\_mail.InnerText = work.email;

cell\_phone.InnerText = work.phone;

del\_btn.InnerHTML = ‘&times’;

cell\_del\_btn.appendChild(del\_btn);

tbl\_block.appendChild(cell\_name);

tbl\_block.appendChild(cell\_surname);

tbl\_block.appendChild(cell\_mail);

tbl\_block.appendChild(cell\_phone);

tbl\_block.appendChild(cell\_del\_btn);

}

});

});

};

Переменной ***sql***присваивается запрос, а затем он отправляется. В переменной ***result***сохраняется результат запроса, т.е. наша таблица из базы данных. В переменной ***n*** сохраняется количество записей, прочитанных из базы. Далее формируется ***html***таблица из блоков ***div***с полученными из базы данными. Создание метода ***update()***. Этот метод служит для изменения записей в базе данных. В метод передаются четыре параметра:

1. name - имя таблицы;
2. field - имя изменяемого поля;
3. value - новое значение изменяемого поля;
4. id - номер строки в базе данных.

DataBase.prototype.update = function(name, field, value, id){

var sql = ‘UPDATE ${name} SET ${field} =’${value}’ WHERE id = ${id}’;

this.db.transaction(function(tx){

tx.exeuteSql(sql);

});

};

В переменной ***sql***формируется запрос, а затем отправляется. Создается метод ***del()***. Этот метод удаляет указанную строку из базы данных. В метод передадется два параметра:

1. name - имя таблицы;
2. id - номер удаляемой строки.

DataBase.prototype.del = function(name, id){

var sql = ‘DELETE FROM ${name} WHERE id =’ + id;

this.db.transaction((tx) =>{tx.executeSql(sql);});

};

В переменной ***sql***формируется запрос, а затем отправляется. Создается метод ***clear()*** - этот метод удаляет таблицу из базы данных. В метод передается в качестве параметра имя удаляемой таблицы.

DataBase.prototype.clear = function(name){

var sql = ‘DROP TABLE ${name}’;

this.db.transaction(function(tx){

tx.executeSql(sql);

});

Document.querySelector(‘.table’).innerHTML = ‘ ‘;

};

В переменной ***sql***формируется запрос, затем отправляется.

Переход к файлу ***app.js***, в котором создается объект ***App()***.

9. Создать конструктор объекта

function App(baseame, tableName){

this.db = new DataBase(baseName);

this.fields = {

addName: document.querySelector(‘#add-name’),

addSurname: document.querySelector(‘#add-surname’),

addMail: document.querySelector(‘#add-mail’),

addPhoneNumber: document.querySelector(‘#add-phone-number’)

};

this.btn = {

add: document.querySelector(‘#add’),

clear: document.querySelector(‘#btnClear’),

del: document.querySelector(‘.table’),

};

};

В конструктор передаются два параметра:

* baseName - имя базы данных;
* tableName - имя таблицы.

Свойству ***db***присваивается экземпляр объекта ***DataBase()***, свойству ***tbl*** присваивается имя таблицы. В объекте ***fields***хранятся ссылки на текстовые поля. В объекте ***btn***хранятся ссылки на кнопки и на блок с таблицей. Создается метод ***create()***. Этот метод создает события.

App.prtotype.create = function(){

this.db.create(this.tbl);

this.btn.add.addEventListener(‘click’, e =>{

if(e.target.tagName == ‘INPUT’){

if(this.fields.addName != ‘ ’ && this.fields.addSurname != ‘ ’ && this.fields.addMail != ‘ ’ && this.fields.addPhoneNumber != ‘ ’){

this.db.add(this.tbl);

this.db.load(this.tbl);

}

for(const key in this.fields){

this.fields[key].value = ‘ ‘;

}

}

});

this.btn.clear.addEventListener(‘click’, e =>{

if(e.target.tagName == ‘BUTTON’){

this.db.clear(this.tbl);

}

});

this.btn.del.addEventListener(‘click’, e =>{

if(e.target.tagName == ‘BUTTON’){

this.db.del(this.tbl, parseInt(e.target.getAttribute(‘data-id’)));

this.db.load(this.tbl);

}

});

this.btn.del.addEventListener(‘dbclick’, e =>{

if(e.target.tagName == DIV’){

e.target.setAttribute(‘conteneditable’, ‘true’);

e.target.focus();

}

});

this.btn.del.addEventListener(‘keydown’, e =>{

if(e.target.tagName == ‘DIV’ && e.target.getAttribute(‘contenteditable’) == ‘true’){

if(e.keyCode == 13){

e.target.setAttribute(‘contenteditable’, ‘false’);

var fld = e.target.getAttribute(‘data-field);

var vle = e.target.InnerText;

var id = parseInt(e.target.getAttribute(‘data-id));

this.db.update(this.tbl, fld, vle, id);

}

}

});

};

В первую очередь вызывается метод***create()*** объекта ***DataBase()***,создается таблица, если она отсутствует. Затем вешается на кнопку добавить событие ***click***. Проверяется, что клик был сделан по кнопке и проверяется. заполнены ли поля. Если всё так, то добавляется новая запись в базу и сразу выводятся обновленные данные в таблицу. После очищаются текстовые поля, проходя по ним в цикле ***for***. Следующим шагом вешается событие ***click***на таблицу и проверяется, что клик произведен по кнопке. Если все верно, то удаляется таблица из базы данных. Теперь вешается на таблицу событие ***click***, проверяется что кликнуто по кнопке, а не по ячейке и. если все верно, удаляется строка из базы данных и загружаются обновленные данные. Далее вешается на таблицу событие ***dblclick***и проверяется, что кликнуто дважды по ячейке таблицы, она создается редактируемой путем добавления атрибута ***contenteditable***. установленного в ***true***, и устанавливается фокус на ячейке. На таблицу вешается событие ***keydown***. Проверяется, вызвано ли событие на ячейки таблицы и является ли ячейка в данный момент редактируемой. Далее проверяется на нажатие кнопки ***ENTER***и, если всё верно, то выключается редактирование ячейки устанавливанием события ***contenteditable***в ***false***. В переменной ***fld***сохраняется имя редактируемого поля, переменной ***vle***присваивается новое значение отредактированного поля, а в переменной ***id***сохраняется порядковый номер строки и обновяются данные в базе.

Создание файла ***main.js***. Повесим на объект ***window***событие ***load***.

window.addEventListener(‘load’, () =>{

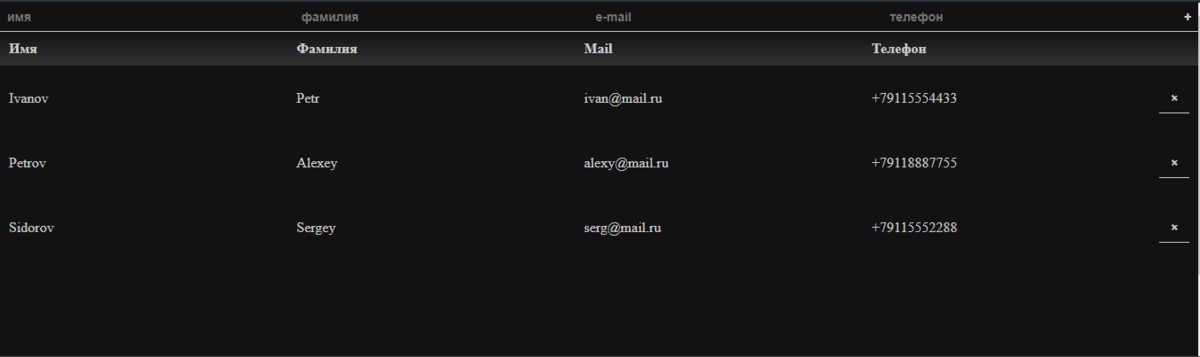
let app = new App(‘dbusr’, ‘users’);

app.db.load(app.tbl);

app.create();

});

В этом событии создадим экземпляр объекта ***App()***. Имя базы данных ***dbusr***, имя таблицы ***users***. Загрузим данные из базы данных. если таковые имеются, и вызовем метод ***create()*** объекта ***App()***. Должно получиться следующее приложение:



внешний вид готового приложения

При этом в панели разработчика база данных имеет следующий вид:

