SQLITE

Instalação do pacote

```
1  npm i react-native-sqlite-storage
```

```
1  npx expo install expo-sqlite
```


https://docs.expo.dev/versions/latest/sdk/sqlite/

SQLITE

Expo SQLite

GitHub npm

`expo-sqlite` gives your app access to a database that can be queried through a WebSQL-like API. The database is persisted across restarts of your app.

Platform Compatibility

Android Device	Android Emulator	iOS Device	iOS Simulator	Web
✅	✅	✅	✅	❌

SQLITE

`SQLite.openDatabase(name, version, description, size, callback)`

Name	Type	Description
name	`string`	Name of the database file to open.
version (optional)	`string`	-
description (optional)	`string`	-
size (optional)	`number`	-
callback (optional)	`(db: WebSQLDatabase) => void`	-

Open a database, creating it if it doesn't exist, and return a `Database` object. On disk, the database will be created under the app's documents directory, i.e. `${FileSystem.documentDirectory}/SQLite/${name}`.

ⓘ The `version`, `description` and `size` arguments are ignored, but are accepted by the function for compatibility with the WebSQL specification.

Returns

↳ `WebSQLDatabase`

SQLITE

Conexão com a base de dados

```
1  //database connection
2  const db = SQLite.openDatabase({
3          name: 'mydb',
4          location: 'default'
5      },
6      () => {
7          console.log("Database connected!")
8      }, //on success
9      error => console.log("Database error", error) //on error
10 )
```

```
const db = SQLite.openDatabase("pessoa.db");
```

SQLITE

`executeSql(sqlStatement, args, callback, errorCallback)`

Name	Type	Description
sqlStatement	`string`	A string containing a database query to execute expressed as SQL. The string may contain `?` placeholders, with values to be substituted listed in the `arguments` parameter.
args (optional)	`(null | string | number) []`	An array of values (numbers, strings or nulls) to substitute for `?` placeholders in the SQL statement.
callback (optional)	`SQLStatementCallback`	Called when the query is successfully completed during the transaction. Takes two parameters: the transaction itself, and a `ResultSet` object (see below) with the results of the query.
errorCallback (optional)	`SQLStatementErrorCallback`	Called if an error occurred executing this particular query in the transaction. Takes two parameters: the transaction itself, and the error object.

Enqueue a SQL statement to execute in the transaction. Authors are strongly recommended to make use of the `?` placeholder feature of the method to avoid against SQL injection attacks, and to never construct SQL statements on the fly.

Returns

↳ `void`

SQLITE

Criar a tabela

```
1  useEffect(() => {
2      createUserTable(); //call create table function here
3  })
4  //create table function
5  const createUserTable = () => {
6      db.executeSql("CREATE TABLE IF NOT EXISTS users (id INTEGER PRIMARY KEY AUTO
7          console.log("Table created successfully");
8      }, (error) => {
9          console.log("Create table error", error)
10     })
11 }
```

```
db.transaction((tx) => {
    tx.executeSql("create table " +
      "if not exists pessoa (indice INTEGER PRIMARY KEY AUTOINCREMENT, " +
      " nome text, telefone text);");
  });
```

SQLITE

CRUD.1

```
//insert a new user record
const createUser = () => {
    let sql = "INSERT INTO users (email, name) VALUES (?, ?)";
    let params = ["yoursocialmd@gmail.com", "MD Sarfaraj"]; //storing user data
    db.executeSql(sql, params, (result) => {
        Alert.alert("Success", "User created successfully.");
    }, (error) => {
        console.log("Create user error", error);
    });
}
```

```
salvarUsuario = () => {
    db.transaction(
      (tx) => {
        tx.executeSql('INSERT INTO pessoa (nome, telefone) VALUES (?,?)',
          [nome, telefone], (resultSet) => {
            Alert.alert("Alerta", "Registro salvo com sucesso");
          }, (error) => {
            console.log(error);
          }
        )
      }
    );
    setNome('');
    setTelefone('');
  };
```

SQLITE

CRUD.2

```
1  //update user record
2  const updateUser = () => {
3      let sql = 'UPDATE users SET email = ?, name = ? WHERE id = ?';
4      let params = ['yoursocialmd@gmail.com', "Mohammad Sarfaraj", 1];
5      db.executeSql(sql, params, (resultSet) => {
6          Alert.alert("Success", "Record updated successfully");
7      }, (error) => {
8          console.log(error);
9      });
10 }
```

SQLITE

CRUD.3

```
1  //delete user record
2  const deleteUser = () => {
3      let sql = "DELETE FROM users WHERE id = ?";
4      let params = [1];
5      db.executeSql(sql, params, (resultSet) => {
6          Alert.alert("Success", "User deleted successfully");
7      }, (error) => {
8          console.log("Delete user error", error);
9      })
10 }
```

SQLITE

CRUD.4

```
1  //list all the users
2  const listUsers = async () => {
3      let sql = "SELECT * FROM users";
4      db.transaction((tx) => {
5          tx.executeSql(sql, [], (tx, resultSet) => {
6              var length = resultSet.rows.length;
7              for (var i = 0; i < length; i++) {
8                  console.log(resultSet.rows.item(i));
9              }
10         }, (error) => {
11             console.log("List user error", error);
12         })
13     })
14 }
```

SQLITE

Complete o código, de acordo com a atividade lançada no Moodle.