



Day 3: Authentication & Introduction to SQL

Re:Coded - Server Side Development Workshop



Agenda, Review, Housekeeping

Today we will implement Authentication in our app, and replace our data mocks with SQLite implementations.

- 1.) Require authorization to access restricted content using Middleware.
- 2.) Create a SQLite Database to store users & use it from our app to finish implementing sign-in and sign-up.
- 3.) Implement the remaining data mocks.



Review



Authentication & Authorization

Express-Session, PassportJS

Can we "hack" our own site and bypass the sign in screen? How will we fix this? What is a sign-in?

Let's fix this problem.

How will we know, once someone "signs in", that it's "the same person" when they request another page?



Sessions

```
npm install express-session
```

<https://www.npmjs.com/package/express-session>



Sessions

```
var expressSession = require('express-session')({  
  secret: 're:coded',  
  resave: false,  
  saveUninitialized: false  
});  
  
app.use(expressSession);
```

خطر!



PassportJS

```
npm install passport
```



PassportJS

```
passport.serializeUser((deserialized_user, cb) => {
  cb(null, deserialized_user.id);
});

passport.deserializeUser((serialized_user, cb) => {
  datasource.get(serialized_user, (user) => {
    cb(null, user);
  });
});

app.use(passport.initialize());
app.use(passport.session());
```




Authorization Middleware

```
var express = require('express');
var router = express.Router();

router.use('/', require('./index'));
router.use('/users', require('./users'));
router.use('/posts', require('./posts'));

module.exports = router;
```



Authorization Middleware

```
app.use(require('./routes/routes.js'));
```



Authorization Middleware

```
npm install connect-ensure-login
```

<https://www.npmjs.com/package/connect-ensure-login>



Authorization Middleware

```
var express = require('express');
var router = express.Router();
var protected = require('connect-ensure-login').ensureLoggedIn('/');

router.use('/', require('./index'));
router.use('/users', require('./users'));
router.use('/posts', protected, require('./posts'));

module.exports = router;
```



SQL

SQLite, DB Browser, sqlite3

How will we store users when they sign up?
Why not just use a JSON file? (There are good reasons)

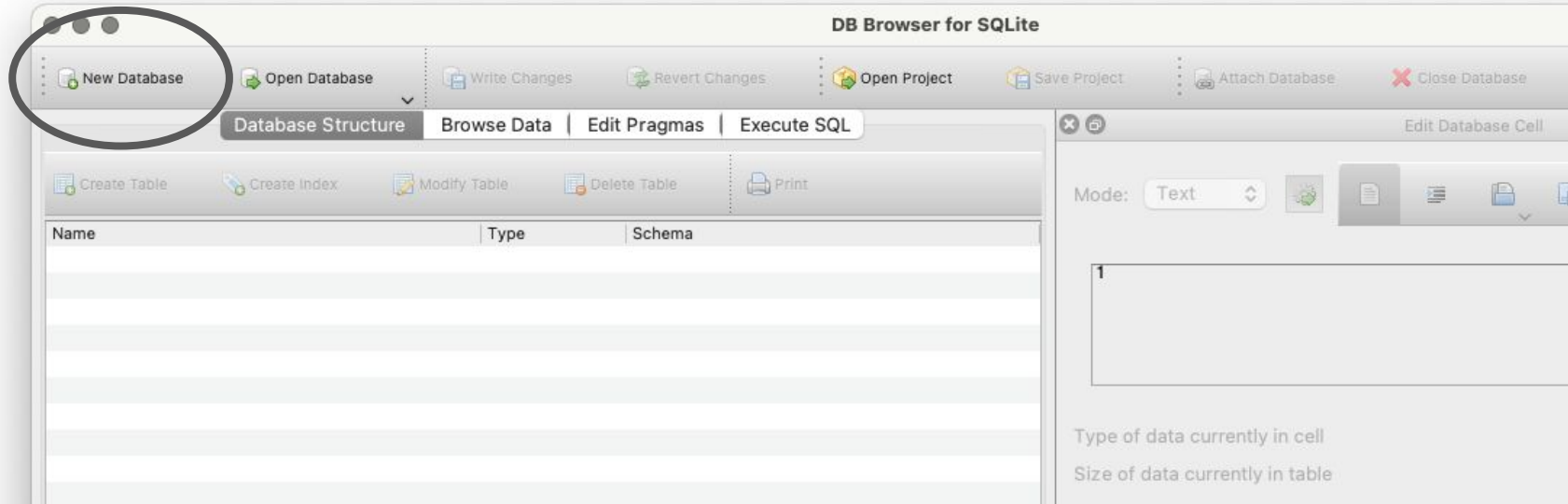


Download DB Browser for SQLite*

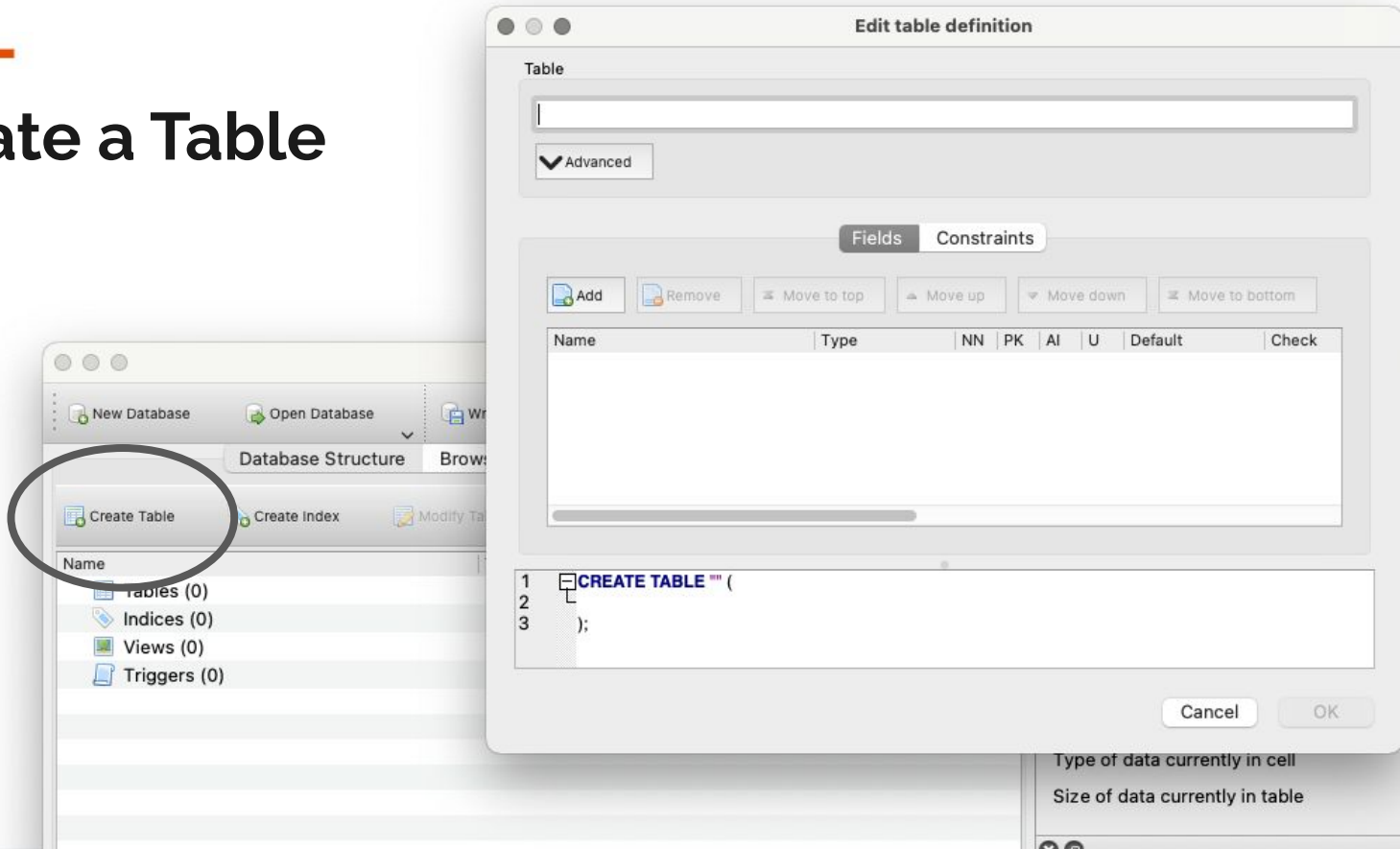
<https://sqlitebrowser.org/dl/>

* As we discussed; we wouldn't use SQLite in production, but this saves us from worrying about setting up database servers locally, and the parts we are interested in are prettymuch the same between all of them.

Create a Database File



Create a Table



Create a Table

- Table names are usually plural and rows represent entities. We will discuss other types of tables tomorrow.
- We almost always want a "Primary Key" - a value that uniquely identifies each row in the table. It should be non-null, auto-incrementing, and unique, as well as marked as the primary key.
- Common names for primary keys include: `id`, `tableName_id`.

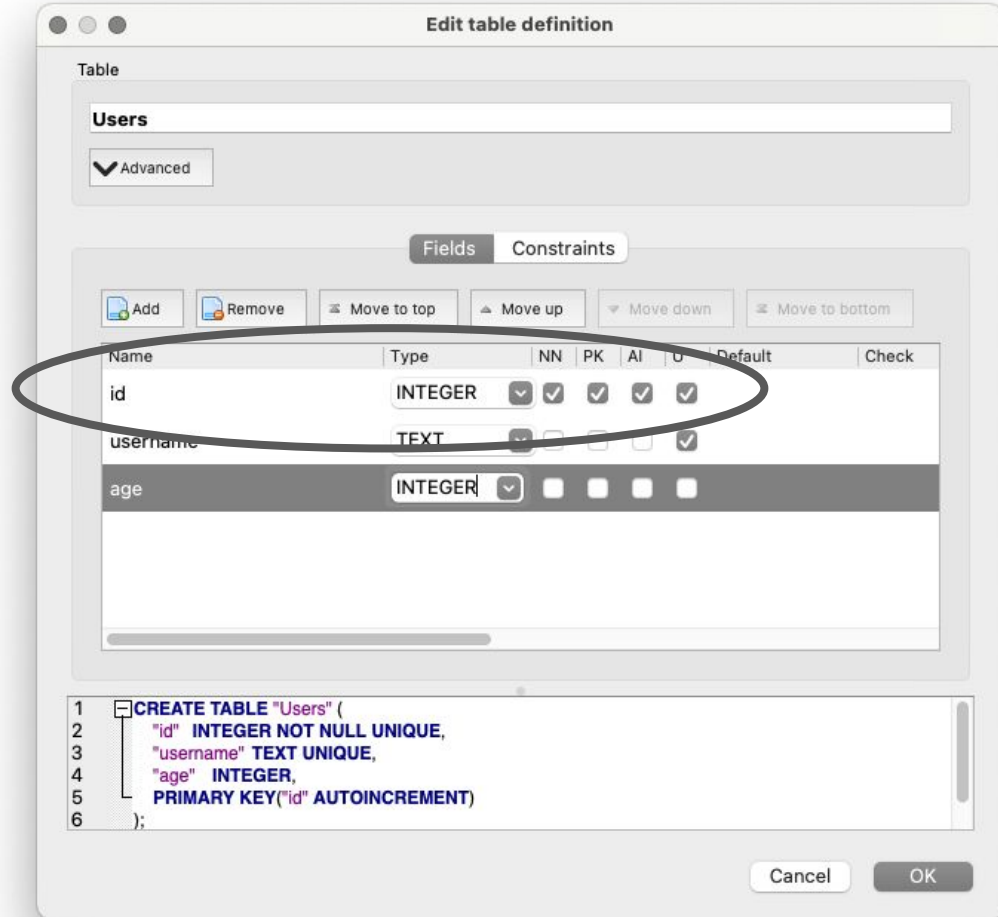


Table: **Users**

Advanced

Fields Constraints

Add Remove Move to top Move up Move down Move to bottom

Name	Type	NN	PK	AI	U	Default	Check
id	INTEGER	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>		
username	TEXT	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>		
age	INTEGER	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		

```
1 CREATE TABLE "Users" (  
2   "id" INTEGER NOT NULL UNIQUE,  
3   "username" TEXT UNIQUE,  
4   "age" INTEGER,  
5   PRIMARY KEY("id" AUTOINCREMENT)  
6 );
```

Cancel OK

Add Columns

- What data do you need to store?
- What is a suitable data type for your new column / field?
- We will discuss constraints, foreign keys, and indices tomorrow.

Table: **Users**

Advanced

Fields Constraints

Add Remove Move to top Move up Move down Move to bottom

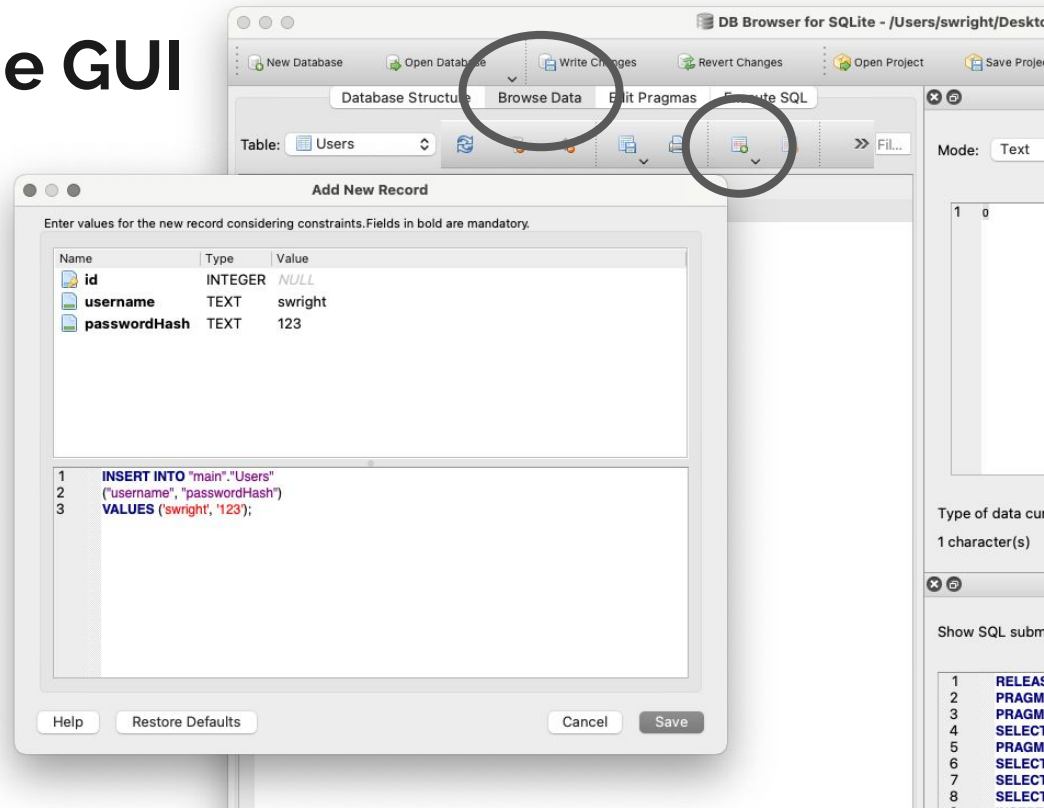
Name	Type	NN	PK	AI	U	Default	Check
id	INTEGER	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>		
username	TEXT	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>		
age	INTEGER	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		

```
1 CREATE TABLE "Users" (  
2   "id" INTEGER NOT NULL UNIQUE,  
3   "username" TEXT UNIQUE,  
4   "age" INTEGER,  
5   PRIMARY KEY("id" AUTOINCREMENT)  
6 );
```

Cancel OK

Manipulate Data in the GUI

- This program can be used to find example SQL queries, as in this example of the SQL for an **INSERT** statement...





CRUD Cheat Sheet

	HTTP Verb	SQL Command
C: Create	POST	INSERT
R: Read/Retrieve	GET	SELECT
U: Update	PUT *	UPDATE
D: Delete	DELETE *	DELETE

* Note that due to real-world problems, endpoints that perform PUT/DELETE-type operations are often represented as POST endpoints instead.



SELECT & WHERE

SELECT [FIELDS] FROM [TABLE] (WHERE [expression])

SELECT * FROM Users

SELECT username FROM Users

SELECT * FROM Users WHERE id=10

SELECT id, 'swright' AS username FROM Users WHERE id = 10

SELECT id, username FROM Users WHERE id = 10 AND (username = "swright" OR username = "steve")



INSERT

```
INSERT INTO Users (username) VALUES ("swright")
```

```
INSERT INTO Users (username, passwordHash) VALUES ("swright", "123")
```



UPDATE

```
UPDATE Users SET username = "steve"
```

```
UPDATE Users SET username = "steve" WHERE id = 10
```

```
UPDATE Users SET username = "swright", passwordHash = "1234" WHERE id = 10
```

```
UPDATE Users SET username = "steve" WHERE username = "swright"
```



DELETE

DELETE FROM Users

DELETE FROM Users WHERE id = 10

DELETE FROM Users WHERE username



WHERE

=, >, <, >=, <=, <>: Do what you'd expect (note that SQL does not use the now-more-popular != operator)

LIKE: String comparison that includes wildcards for "a single character" () and "any number of characters" (%)

```
SELECT * FROM Users WHERE username LIKE "_wright"
```

```
SELECT * FROM Users WHERE username LIKE "*righ*"
```

IN: Value comparison similar to (x = o1 OR x = o2 OR x = o3) (More on this tomorrow)

```
SELECT * FROM Users WHERE id IN (10, 20, 30, 40)
```



Using SQLite

```
npm install sqlite3
```

<https://www.npmjs.com/package/sqlite3>



Using SQLite

```
var sqlite3 = require('sqlite3');  
var db = new sqlite3.Database('database.db');
```



Using SQLite

	Used when...
db.run	You don't expect or need data back (UPDATE, INSERT, DELETE statements)
db.get	You only expect or want the first row (SELECT)
db.all	You expect more than one row and you want an array
db.each	You expect more than one row and you want to process them one at a time



Querying

```
db.get("SELECT * FROM Users WHERE username = ?",  
      [ username ],  
      (err, user_row) => {  
    var id = user_row.id;  
    ...  
  });
```



Debugging

```
db.on('profile', (sql, time) => {  
  console.log(sql);  
});
```

Let's implement the Users data source together and get sign-in and sign-up working!



bcrypt

```
npm install bcrypt
```




Create a Hash

```
bcrypt.hash(credentials.password, saltRounds, (err, passwordHash) => { });
```



Verifying a Password

```
bcrypt.compare(credentials.password,  
               user_row.passwordHash,  
               (err, passwords_match) => { });
```

تحدث معي على سلاك!

Now you try...

Please create an implementation of the Posts data source that uses SQLite.

- Create a table(s) as needed.
- Implement a data source - you will have to write SQL queries.

If you finish early:

- If you write multiple paragraphs in your post, does it display as you expect? Why? Can you fix it in your branch?
- Try implementing upvoting.



URL Parameters

```
router.get('/:id', (req, res, next) => {  
  var id = req.params['id'];  
  ...  
});
```



Data Normalization

How did you store the username for a post? Is it a "text" field in the "Posts" table?

What would happen if we allow users to change their usernames? Would our data be consistent?

We could update all these rows.

That may be the best choice if we don't expect users to update their usernames very often, and if we expect to need this data quickly and often.

But let's talk about the "Relational" in "Relational Databases"...



JOINS

```
SELECT Posts.*, Users.username AS author FROM Posts INNER JOIN Users ON Posts.user_id = Users.id
```



See You Tomorrow!