NORTH CODERS

<Spring Data Lecture Nine />

<Recap />

- Covered Java objects and OOD
  - E....
  - A....
  - I......
  - P.....
- Intro to IoC and Dependency Injection
- Spring MVC
- Sprint ReST
- Your React and Java project

<Now lets introduce data />

We can utilise the JPA (Java Persistence API) to begin working with data.

At this stage we'll focus on relational structured data rather than non-relation data such as NoSQL (Mongo, Dynamo etc)

<We'll cover />

- Extending your REST controllers to fetch data
- Some more externalised config
- Automation of database updates with Flyway
- MySQL and database transactions (ACID)
- A bit of Docker thrown in for good measure

<Entities />

Think of an entity as an object that maps back to your database.

For example the Car object could be though of as an entity that maps back to the 'Car' table in the database.

<Car.java (as an entity) />

```java
package com.northcoders.model;

import javax.persistence.*;

@Entity
public class Car {

    private Long carId = null;
    private String carMake = null;
    private String carModel = null;
    private int engineSize = 0;

    @Id @GeneratedValue(strategy = GenerationType.AUTO)
    @Column(name="car_id")
    public Long getCarId() {
        return carId;
    }

    public void setCarId(Long carId) {
        this.carId = carId;
    }

    @Column(name="car_make")
    public String getCarMake() {
        return carMake;
    }

    ...
}
```

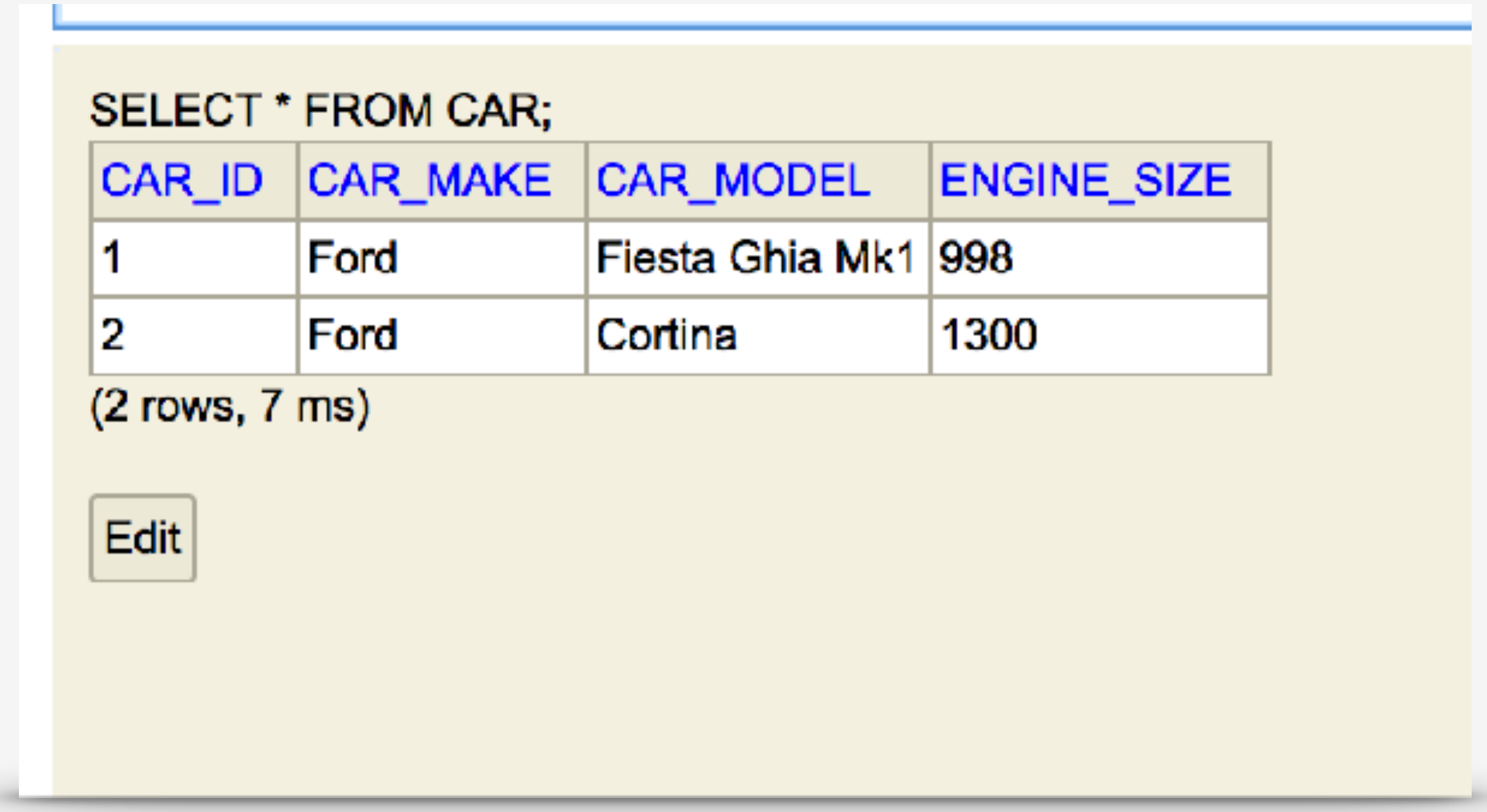
<Car.java (as an entity) />

```
package com.northcoders.model;

import javax.persistence.*;

@Entity
public class Car {

    private Long carId = null;
    private String carMake = null;
    private String carModel = null;
    private int engineSize = 0;

    @Id @GeneratedValue(strategy = GenerationType.AUTO)
    @Column(name="car_id")
    public Long getCarId() {
        return carId;
    }

    public void setCarId(Long carId) {
        this.carId = carId;
    }

    @Column(name="car_make")
    public String getCarMake() {
        return carMake;
    }

    ...
}
```

<Car.java (as an entity) />

```
package com.northcoders.model;

import javax.persistence.*;

@Entity
public class Car {

    private Long carId = null;
    private String carMake = null;
    private String carModel = null;
    private int engineSize = 0;

    @Id @GeneratedValue(strategy = GenerationType.AUTO)
    @Column(name="car_id")
    public Long getCarId() {
        return carId;
    }

    public void setCarId(Long carId) {
        this.carId = carId;
    }

    @Column(name="car_make")
    public String getCarMake() {
        return carMake;
    }

    ...
}
```

<And on the database... />



SELECT * FROM CAR;

CAR_ID	CAR_MAKE	CAR_MODEL	ENGINE_SIZE
1	Ford	Fiesta Ghia Mk1	998
2	Ford	Cortina	1300

(2 rows, 7 ms)

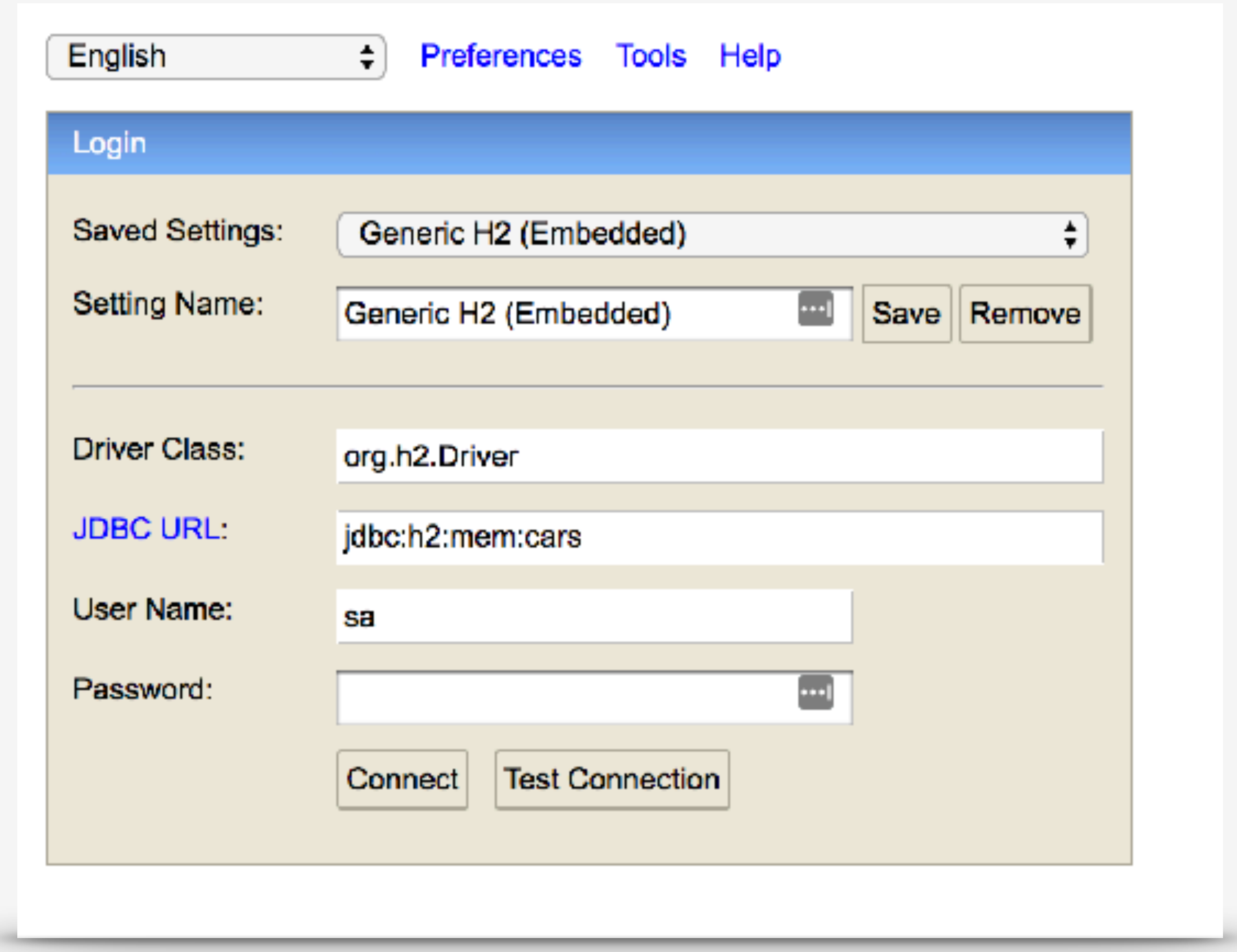
<H2 Database />

This example currently ships with the inbuilt H2 database.

It acts and feels like a MySQL database

<H2 Database />

- http://localhost:8080/h2-console



Saved Settings: Generic H2 (Embedded)

Setting Name: Generic H2 (Embedded)

Driver Class: org.h2.Driver

JDBC URL: jdbc:h2:mem:cars

User Name: sa

Password:

<Repositories />

Think of these as your way of providing access to your data.

< CarRepository />

```
package com.northcoders.repository;

import com.northcoders.model.Car;
import org.springframework.data.repository.CrudRepository;

public interface CarRepository extends CrudRepository<Car, Long> {
}
```

<WTF - Theres no methods />

- Extends CrudRepository
- AutoMagically
- Java Generics

<Java Generics />

```
/**
 * Generic version of the Box class.
 * @param <T> the type of the value being boxed
 */
public class Box<T> {
    // T stands for "Type"
    private T t;

    public void set(T t) { this.t = t; }
    public T get() { return t; }
}
```