COMP 4513 Assignment #1

Due Monday October 20th at midnight
Version 1.0, Sept 23

Overview

This assignment provides you with an opportunity to create an API in Node. You will also be required to provision a database on supabase.

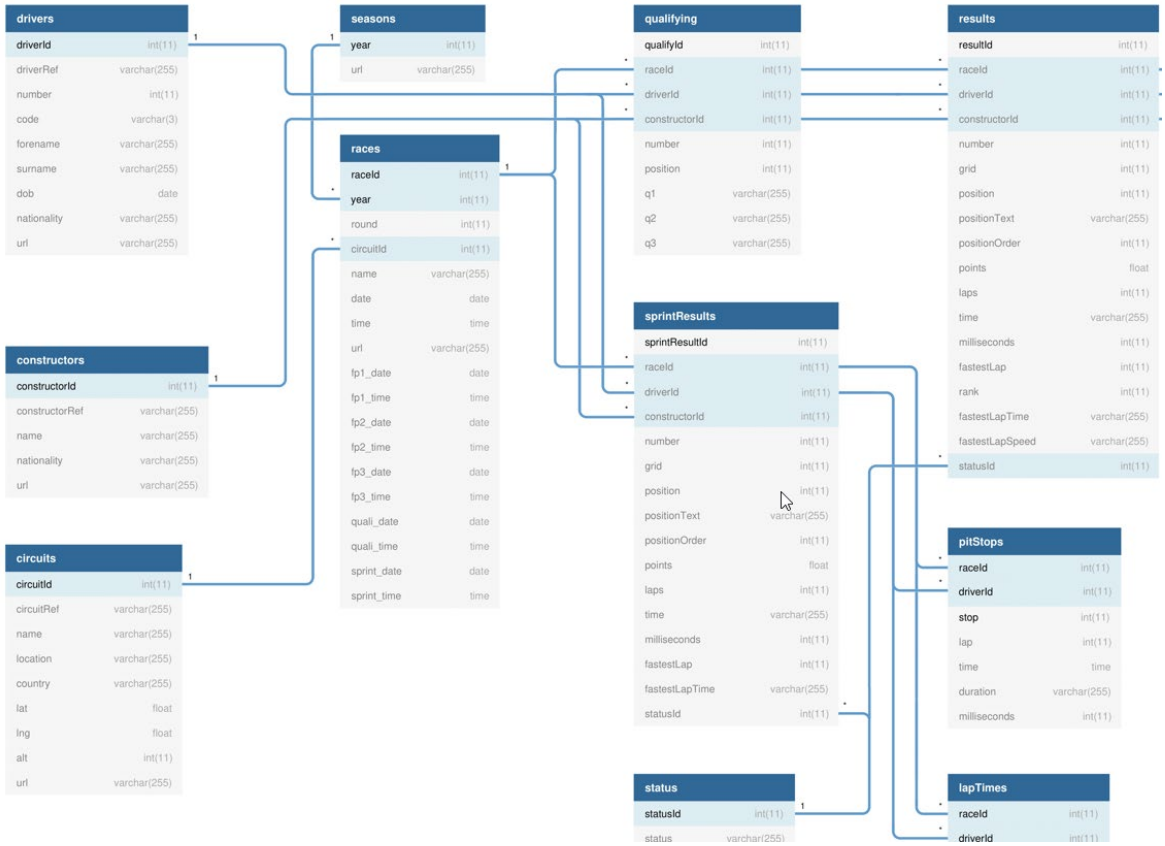
The data you have been provided with contains Formula 1 car racing data. In this assignment, you must implement a variety of Web APIs in Node. These APIs will provide the user with the ability to examine races within a single season and examine their results, drivers, and constructors.

Grading

The grade for this assignment will be broken down as follows:

Programming Design and Documentation	15%
Hosting + Correct readme	10%
Functionality (follows requirements)	75%

Database



Data has been provided as an SQLite database and as a variety of csv files. The database consists of quite a few tables: you are only interested in this assignment in some of them:

- **seasons**: list of seasons covered (theoretically) in the data files.
- **circuits**: A race happens on a circuit. A circuit is used 0 or 1 times a season.
- **constructors**: Each constructor usually has two cars (and thus two drivers) in each race. Fans care about the drivers and the constructors.
- **drivers**: Each car has a single driver. Each driver in a season drives a single car.
- **races**: In a season there are a certain number of races (in 2022 there were 22 races). For your queries on this table, just hard code the `year` field value to 2022.
- **qualifying**. Before the race there is qualifying, which consists of three rounds. The five slowest drivers get knocked out of the next round, so only 10 drivers will have q1, q2, and q3 times. The qualifying position determines the race starting positions.
- **results**: A single race has a series of results, one for each driver. The drivers and constructors earn points for their results. Race followers are especially interested in a driver's results, which includes not just their finish position, but how long they took, their fastest lap, and so on.

This data theoretically has decades of race data, but unfortunately, quite a lot of data was lost in the exporting process. As a result, when testing some your queries, stick with data that falls within the years/seasons 2019-2023.

The tables are organized with primary keys and foreign keys so you will need to use `INNER JOINS` where necessary.

Database Choice

You have two choices as to how you approach the database retrieval in this assignment.

1. Use the provided sqlite database. As you have seen in Lab14b exercises 1-4, sqlite is a file-based SQL database that requires using some type of API for running SQL queries. The lab uses the sqlite3 package. There is also a built-in sqlite package called node:sqlite which you are welcome to use instead if that is your preference.
2. Use a cloud-based DBMS. Lab14b exercises 5-9 uses supabase. The disadvantage to supabase is that it has its own learning curve and you can't "see" or easily modify the SQL generated by the supabase API. The advantages are twofold: a) you can put "practical experience with a cloud database" on your resume, which is actually a pretty desirable attribute in a entry-level developer, and b) when you create your assign2 in react, it will considerably simplify your hosting requirements, since you won't need to maintain an external node host to run your web API; instead, you will be able to use the supabase API directly in your React application (as in exercise 9 in Lab14b).

Submitting and Hosting

You will be using Node in this assignment. This will mean your assignment will need to reside on a working host server. Static hosts (e.g., github pages) will not work.

For this assignment, I would recommend using either glitch.com or render.com, both of which provides a free option for hosting Node applications. Do note that these free projects go to sleep after a set period of inactivity, so be aware that the first request of a slept hosted node application will take some time to awaken.

It is possible that there are other node hosting options which are superior to glitch or render. Feel free to use whatever hosting environment you like.

When your hosting is working and the assignment is ready to be marked, then send me an email with the following information:

- The URL of the github repo so that I can mark the source code. If your repo is private, then add me as a collaborator.
- In the `readme.md` file for your repo, provide links to the APIs on glitch/render so I can test them (see details below).

NOTE: Free tier supabase projects will go inactive after one week of inactivity (and thus your API won't work). Please login into supabase and run a test query every few days after the due date so this doesn't happen!!

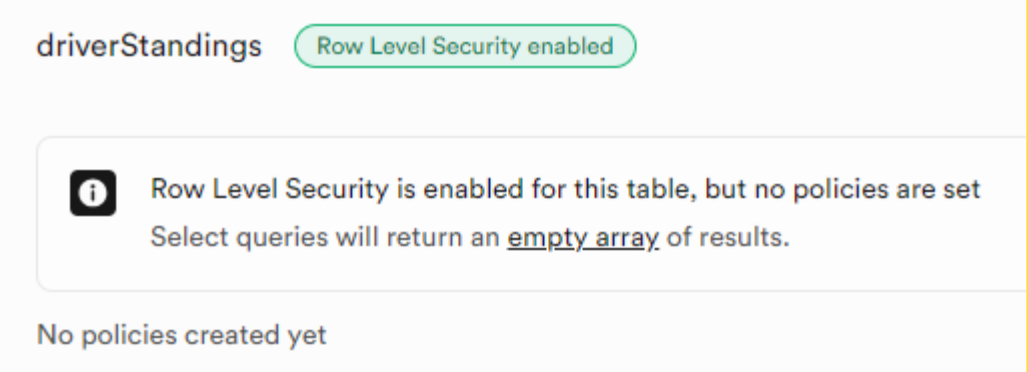
Recommended Workflow

I recommend you approach this assignment in the following order:

1. Complete Lab14b
2. Set up a github repo for your source code.
3. Implement the SQL for each of the API routes using the SQL Editor in supabase or one of the sqlite tools available. Save the SQL query text in a file. While this step is not strictly necessary, it will make step 5 easier.
4. Compare your query results to someone else's query results. Again, this isn't necessary, but it might help you catch errors in your query logic for the more complex queries.
5. Implement the APIs in Express. If you have created the SQL queries first, then you can simply convert them to supabase's query builder syntax or sqlite code.
6. Set up the hosting. The hosting might take longer to set up then you anticipate, so be sure to leave ample time for it. Some students found glith to be awful, but some found it easy to use; the same goes for render. Be willing to post questions on the class discord if you have issues with your hosting; be willing to answer others' questions!
7. Construct a `readme.md` file in your github repo with the expected example API request links (see page 5).
8. **Test the link APIs in the readme file after hosting!!!**
9. Send me an email with the required info (see page 2).

Common Supabase Problems

1. API returns a blank screen. Evidently error messages are for rookies and not for supabase users. Generally this happens because there is a problem in your SQL (i.e., your field list, your filter, or your modifier). Fix it! Check for field names being incorrectly spelled or that have the wrong case. Are you including a field from a table in which relations haven't been set? **Look for trailing commas as they will mess it up.**
2. API returns an empty array. You likely haven't set a row-level security policy that allows read only access for one of the tables. For instance (ignore the table name, this is from last year):



3. You are getting results back from supabase that shouldn't be returned based on the filters. For instance, instead of a full object, you will see a **null** value instead. You will see this when you are trying to use a filter on a joined table, e.g.,

```
.select(`raceId,year, circuits (circuitRef,name)`)
.eq('circuits.circuitRef','monza')
```

Strangely (at least to me), supabase seems to default to a FULL JOIN rather than an INNER JOIN in this situation. For instance, let's say you want to see races and circuits data for races at monza. When it does a FULL JOIN, you will get not only races at monza, but all races not at monza. But because the filter is in place, the races not at monza will have a circuits value of null.

To fix this, you need to tell supabase to make the join an inner join via:
`.select(`raceId,year, circuits!inner (circuitRef,name)`)`

4. Your sort isn't working or is resulting in the dreaded blank screen. To sort on a field in a linked table, you have to add in a `referencedTable` property, e.g.,
`.order('year', { referencedTable: 'races', ascending: false })`

API Functionality

You must create the following APIs with the specified routes and functionality. The returned data must be JSON format.

`/api/circuits`	Returns all the circuits
`/api/circuits/`*`ref`*	Returns just the specified circuit (use the `circuitRef` field), e.g., `/api/circuits/monaco`
`/api/circuits/season/`*`year`*	Returns the circuits used in a given season (order by round in ascending order), e.g., `/api/circuits/season/2020`
`/api/constructors`	Returns all the constructors
`/api/constructors/`*`ref`*	Returns just the specified constructor (use the `constructorRef` field), e.g., `/api/constructors/mclaren`
`/api/drivers`	Returns all the drivers
`/api/drivers/`*`ref`*	Returns just the specified driver (use the `driverRef` field), e.g., `/api/drivers/hamilton`
`/api/drivers/search/`*`substring`*	Returns the drivers whose surname (case insensitive) begins with the provided substring, e.g., `/api/drivers/search/sch`
`/api/drivers/race/`*`raceId`*	Returns the drivers within a given race, e.g., `/api/drivers/race/1106`
`/api/races/`*`raceId`*	Returns just the specified race. Don't provide the foreign key for the circuit; instead provide the circuit name, location, and country.
`/api/races/season/`*`year`*	Returns the races within a given season ordered by `round`, e.g., `/api/races/season/2020`
`/api/races/season/`*`year`*`/`*`round`*	Returns a specific race within a given season specified by the round number, e.g., to return the 4th race in the 2022 season: `/api/races/season/2022/4`
`/api/races/circuits/`*`ref`*	Returns all the races for a given circuit (use the `circuitRef` field), ordered by `year`, e.g. `/api/races/circuits/monza`

/api/races/circuits/*ref*/season/*start*/*end*	Returns all the races for a given circuit between two years (include the races in the provided years), e.g., /api/races/circuits/monza/season/2015/2020 /api/races/circuits/monza/season/2020/2020
/api/results/*raceId*	Returns the results for the specified race, e.g., /api/results/1106 Don't provide the foreign keys for the race, driver, and constructor; instead provide the following fields: driver (driverRef, code, forename, surname), race (name, round, year, date), constructor (name, constructorRef, nationality). Sort by the field grid in ascending order (1st place first, 2nd place second, etc).
/api/results/driver/*ref*	Returns all the results for a given driver, e.g., /api/results/driver/max_verstappen
/api/results/drivers/*ref*/seasons/*start*/*end*	Returns all the results for a given driver between two years, e.g., /api/results/drivers/sainz/seasons/2022/2022
/api/qualifying/*raceId*	Returns the qualifying results for the specified race, e.g., /api/qualifying/1106 Provide the same fields as with results for the foreign keys. Sort by the field position in ascending order.
/api/standings/drivers/*raceId*	Returns the current season driver standings table for the specified race, sorted by position in ascending order. Provide the same fields as with results for the driver.
/api/standings/constructors/*raceId*	Returns the current season constructors standings table for the specified race, sorted by position in ascending order. Provide the same fields as with results for the constructor.

For each of the requests that take parameters, your API needs to handle a Not Found condition. For instance, if a ref or year doesn't exist, **return a JSON-formatted error message** that indicates the requested request did not return any data. For the routes with a start and end year, provide a different error message if the end year is earlier than the start year.

GitHub Readme

Remember that potential employers will look at your github repos and will want to see best practices. That is, think of your github repo readme files as if they were part of your resume/portfolio. Here are some excellent sample github readmes from other classes that demonstrate the level of detail that you may want to emulate.

README

Formula 1 Data APi

Overview

This project is an API for querying F1 data - ciruits, constructors, drivers, races and results. The data is returned in Json format

Built with

Node Js - JS runtime

Express - Routing

Glitch - For deployment -https://sophisticated-citrine-savory.glitch.me

API Endpoints

API Endpoint	Description
`/api/circuits`	Get all circuits
`/api/circuits/:id`	Get circuit by ID
`/api/constructors`	Get all constructors
`/api/constructors/:ref`	Get constructor info by reference
`/api/constructorResults/:constructorRef/:year`	Get all constructor results for a specified year
`/api/drivers`	Get all drivers info
`/api/drivers/:ref`	Get driver info by reference
`/api/driverResults/:ref/:year`	Get all race results over a season for the specified driver
`/api/races/season/:year`	Get info about all the races specified by the year/season
`/api/races/id/:id`	Get info about the race specified by id
`/api/results/race/:id`	Get all results for the race specified by id
`/api/results/season/:year`	Get all the race results for every race in the specified season

Test links

- /api/circuits
- /api/circuits/1
- /api/constructors
- /api/constructors/mclaren
- /api/coNSTrucTors/mclaren

README

COMP 3612 (Fall 2023)

Assignment #3: Node API

Overview

This repository contains code for a F1 API. The assignment makes use of Node and Express to efficiently manage the server and the possible routes that may be taken. Data for circuits, constructors, drivers, and results for seasons, constructors, drivers or a combination of season and driver or constructor are able to be fetched from the server. The data is returned in JSON form.

Node.js 22.12.0 Express 4.21.1 Deployed on Render.com

Example:

Request: `/api/drivers/max_verstappen`

Response:


```
{
  "driverId": 830,
  "driverRef": "max_verstappen",
  "number": 33,
  "code": "VER",
  "forename": "Max",
  "surname": "Verstappen",
  "dob": "1997-09-30",
  "nationality": "Dutch",
  "url": "http://en.wikipedia.org/wiki/Max_Verstappen"
}
```


Project Files

File	Description
`F1_API.js`	Contains the code for the server itself and starts listening for requests.
`data_provider.js`	Fetches data from the data folder and exports it for use by the router.
`router.js`	Handles possible routes, filtering and returning appropriate JSON data.

Testing

- Render.com Hosting
- /api/circuits
- /api/circuits/1
- /api/constructors
- /api/constructors/mclaren
- /api/coNSTruCTors/mclaren

Example API Requests

In the `readme.md` file for your assignment repo, you must supply a list of links that allow me to test each of your APIs. Please add the following test links (they must be clickable links) in this file:

/api/circuits	/api/races/season/2021
/api/circuits/monza	/api/races/season/1800
/api/circuits/calgary	/api/races/season/2020/5
/api/constructors	/api/races/season/2020/100
/api/constructors/ferrari	/api/races/circuits/7
/api/drivers	/api/races/circuits/7/season/2015/2022
/api/drivers/Norris	/api/races/circuits/7/season/2022/2022
/api/drivers/norris	/api/results/1106
/api/drivers/connolly	/api/results/driver/max_verstappen
/api/drivers/search/sch	/api/results/driver/connolly
/api/drivers/search/xxxxx	/api/results/drivers/sainz/seasons/2021/2022
/api/drivers/race/1069	/api/results/drivers/sainz/seasons/2035/2022
/api/races/1034	/api/qualifying/1106
	/api/standings/drivers/1120
	/api/standings/constructors/1120
	/api/standings/constructors/asds

Note: you will need to preface the above URLs with the URL of your host. For instance, if your Glitch URL is `https:/smashing-squirrels.glitch.me`, then the URL for the second test link would be `https:/smashing-squirrels.glitch.me/api/paintings/63`