COMP 4513 Assignment #1

Due Monday February 26th at 10am
Version 2.1, Feb. 20, changes in blue

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.

Files

The database is the same f1 database used in Lab14b. You will be able (eventually) to find the csv files as well at the GitHub repo for the assignment:

`https://github.com/mru-comp4513-archive/w2024-assign1`

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%

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. 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 querieis first, then you can simply convert them to supabase's query builder syntax.
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.
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).

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.

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

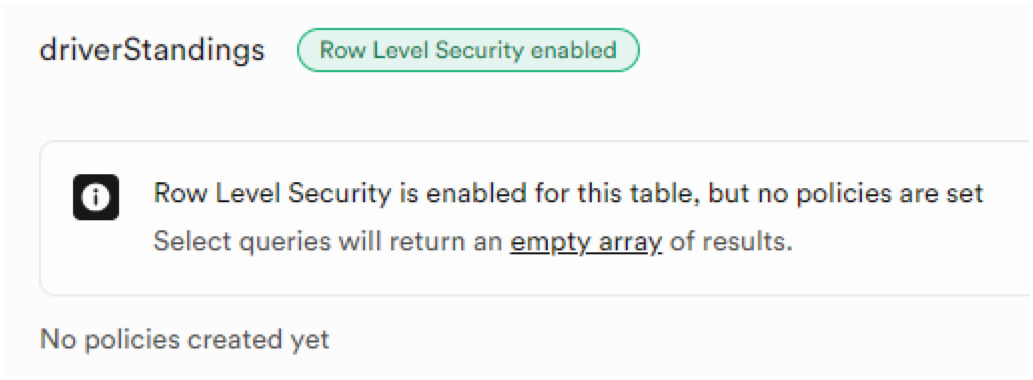
- The URL of the github repo so that I can mark the source code. If your repo is private, then add me as a collaborator.
- In the `readme.md` file for your repo, provide links to the APIs on glitch/render so I can test them (see details below).

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!!

Common Problems

1. API returns a blank screen. This is due to 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:



driverStandings Row Level Security enabled

Row Level Security is enabled for this table, but no policies are set

Select queries will return an empty array of results.

No policies created yet

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/seasons`	Returns the seasons supported by the API (that is, all the data in the `seasons` table).
`/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/constructors/season/`*`year`*~~	~~Returns the constructors within a given season, e.g., `/api/constructors/season/2020`~~
`/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/season/`*`year`*~~	~~Returns the drivers within a given season, e.g., `/api/drivers/season/2022`~~
`/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/driver/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/raceId/drivers`	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/raceId/constructors`	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 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.

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/seasons`	`/api/races/1034`
`/api/circuits`	`/api/races/season/2021`
`/api/circuits/monza`	`/api/races/season/2020/2022`
`/api/circuits/calgary`	`/api/races/season/2022/4`
`/api/constructors`	`/api/races/circuits/monza`
`/api/constructors/ferrari`	`/api/races/circuits/monza/season/2015/2022`
~~`/api/constructors/season/2020`~~	`/api/results/1106`
`/api/drivers`	`/api/results/driver/max_verstappen`
`/api/drivers/Norris`	`/api/results/driver/connolly`
`/api/drivers/norris`	`/api/results/driver/sainz/seasons/2021/2022`
`/api/drivers/connolly`	`/api/qualifying/1106`
`/api/drivers/search/sch`	`/api/standings/1120/drivers`
`/api/drivers/search/xxxxx`	`/api/standings/1120/constructors`
~~`/api/drivers/season/2022`~~	`/api/standings/asds/constructors`
`/api/drivers/race/1069`	

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/433`