Project Participants:

Katherine (Kathy) Moss

Title:

Rocket League Database

Executive Summary:

This project is based on the video game "Rocket League", where players in cars are pitted against each other in a soccer arena, trying to score goals into the opposing team's net with a giant soccer ball.

One player can have many different cars with different qualities. A player can have different ranks (like skill levels), and they can pick up ranks over time (an optional many). The database will differ from the actual game, in that, the cars a player gets will be tied to their rank, thus creating a many-to-many relationship and a joins table.

Initial Features:

- DB Design contains 4 tables (players, cars, ranks_earned, and car_rank)

  - DB must contain at least 3 tables ✔

- Contains 3 Entities (players, cars, ranks_earned – > Players, Cars, RanksEarned as Entity Classes)

  - DB must contain at least 3 entities ✔

- CRUD Ops:

  Players = C, R

  - Player by ID = R, U

  Cars = C, R, U, D

  - Car by ID = R, U, D

  RanksEarned by a specific player by ID= C, R

  RanksEarned by a specific car via ID = C, R

<u>Explanation:</u>

- Players – you can add a new player (C) to the DB and look at all players (R), but you can't remove a player's account … once it's there, they are forever in Rocket League
  - Player by ID
    - view a player by specific ID (R)
    - change aspects of the player like banner, border, and anthem (U) given a specific ID

- Cars – can add a new car (C) or view list of cars (R)
  - Car by ID
    - view a car by specific ID (R)
    - change color and finish (U)
    - remove from inventory (D)

- RanksEarned – You can gather ranks and look at them (CR), but ranks are not removed (unless the car that has the rank is removed child/parent relationship), they're kind of like badges.

  - Each entity should have at least 1 CRUD op ✔

  - 1+ entity has all CRUD op ✔

There are many cars in a player's inventory (optional many) and a player can have many (optional) ranks that they have earned. The car can pick up multiple ranks. A rank can be established for multiple cars. This joins relationship is established in the car_rank table.

- 1+ Many-to-Many relationship ✔

The Many-to-Many relationship between cars + ranks can be…

C – can add a rank to a car

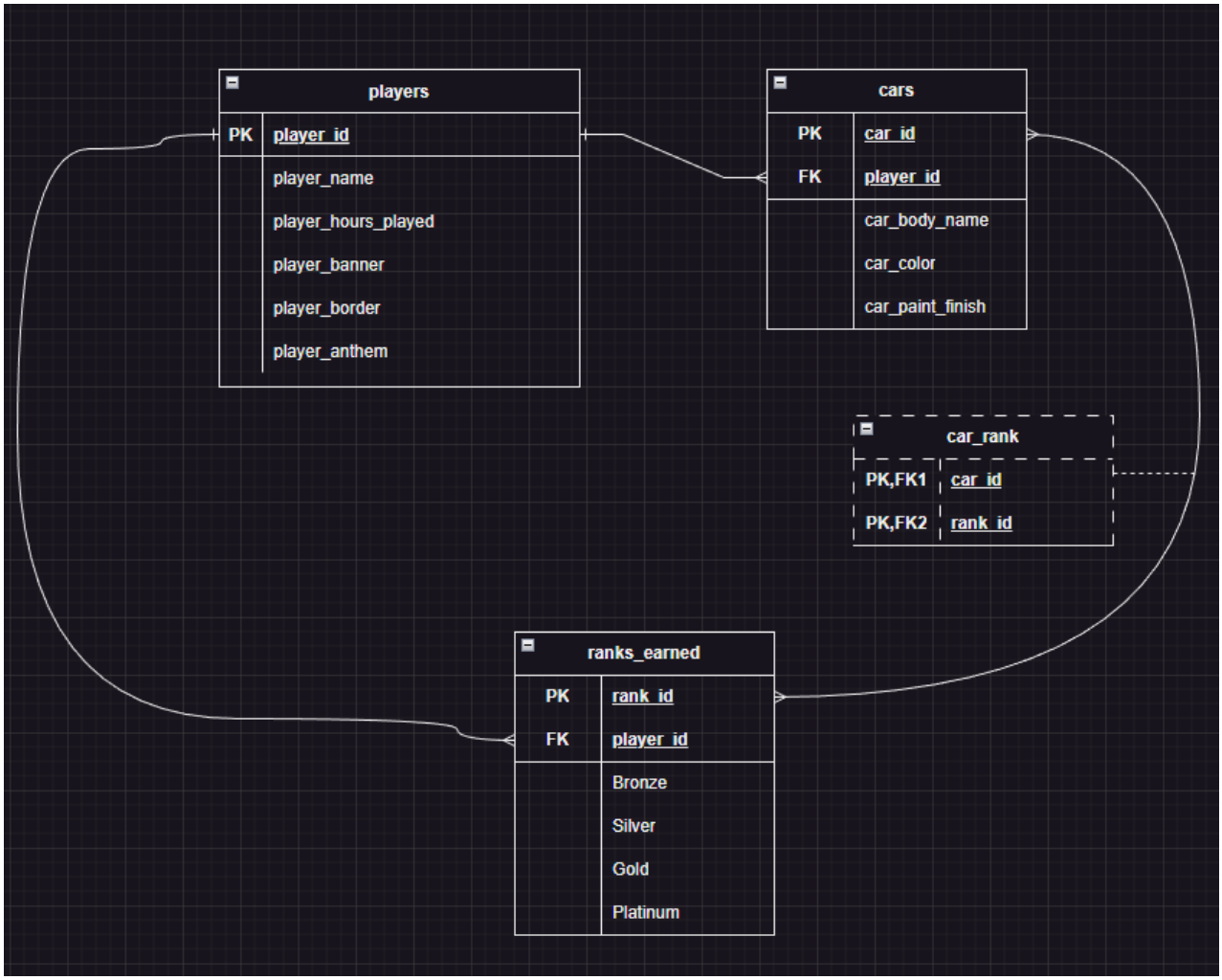
R – See rank on a car

U – Add rank to car

- Many-to-Many relationship with 1+ CRUD op ✔

- REST API Server test through: Swagger, Postman, **ARC** ✔

- Database Design –


players
PK
player_id
player_name
player_hours_played
player_banner
player_border
player_anthem
cars
PK
car_id
FK
player_id
car_body_name
car_color
car_paint_finish
car_rank
PK,FK1
car_id
PK,FK2
rank_id
ranks_earned
PK
rank_id
FK
player_id
Bronze
Silver
Gold
Platinum

Planned Features/API Endpoints:

*** one name to tie endpoints together with REST mapping: /rocketleagueserver**

*** all mappings below are a step below under the /rocketleagueserver**

/players

Verb	Path Parameters	Query Parameters	Request Body	Response
GET (R)	None	None	None	200 – shows list of all players
POST (C)	None	None	player_name; player_hours_played; player_banner; player_border; player_anthem	201 – A new response has been created with the request body in previous column

/players/{player_id}

Verb	Path Parameters	Query Parameters	Request Body	Response
GET (R)	player_id	None	None	200 – shows particular player and attributes
PUT (U)	player_id	None	player_name; player_hours_played; player_banner; player_border; player_anthem	200 – shows player with updated attributes

/players/{player_id}/cars

Verb	Path Parameters	Query Parameters	Request Body	Response
GET	None	None	None	200 – shows list of all cars for a specific player
POST	None	None	car_body_name; car_color; car_paint_finish	201 – A new response has been created with the request body in previous column, added a new car to

				the garage

/cars/{car_id}

Verb	Path Parameters	Query Parameters	Request Body	Response
GET	car_id	None	None	200 – show a specific car by ID with specific attributes
PUT	car_id	None	car_body_name; car_color; car_paint_finish	200 – shows car by ID with updated attributes
DELETE	car_id	None	None	204 – No Content, removed from garage

/players/{player_id}/ranks_earned

Verb	Path Parameters	Query Parameters	Request Body	Response
GET (R)	None	None	None	200 – shows list of all ranks earned by a player
POST (C)	None	None	Any of the following: Bronze, Silver, Gold, Platinum	201 – A new response has been created with the request body in previous column

Ranks earned for a specific car:

***Joins Table Endpoints:**

/cars/{car_id}/ranks_earned

- make sure join table has at least one CRUD op ✔

Verb	Path Parameters	Query Parameters	Request Body	Response
GET (R)	car_id	None	None	200 – shows all ranks earned for specific car
POST (C)	car_id	None	Any of the following but no duplicates for a certain car: Bronze, Silver, Gold, Platinum	201 – Response created, added a rank to a car

// Note: ddl-auto: create --- > done with everything needed for DB? : ddl-auto: update

AKA:

- Entities are correct

- Tables are created properly

=> changing the ddl settings makes sure the app will only update tables and not drop them and create them again

Stretch Goals (to be completed if time allows, or after graduation):

- Make a more detailed video showcasing my work
- Add a README file
- GitIgnore ?
- Add more ranks – > Rocket League has:
  - Unranked, Bronze, Silver, Gold, Platinum, Diamond, Champion, Grand Champion, Supersonic Legend, each with Tiers I, II, III, and IV
- Create a frontend for the backend, which requires:

- getting reacquainted with FE
- learning more React
- making sure everything is optimized for different screens
- knowing how to tie the FE to BE