**QUESTIONS**

1. **Name the top most represented sport in the Olympics**

Let's uncover the most represented sport in the Olympics! Count unique athlete occurrences, group the data by sport, and reveal the top three sports with the highest number of athletes. Ready to dive in? Let's go!

* Select the sport field from the summer\_games table.
* Create athletes by counting the distinct occurrences of athlete\_id.
* Group by the sport field.
* Make the report only show 3 rows, with the highest athletes at the top.

1. **Compare athletes and the events of the above**

Let's explore the relationship between athletes and events in the summer games! By pulling a report from the summer\_games table, we'll discover the unique number of events and athletes for each sport. By grouping the data based on the sport field, we'll gain valuable insights into the diverse sports landscape of the games. Get ready to unveil the fascinating connections between athletes and events! Let's dive in!

* Pull a report that shows each sport, the number of unique events, and the number of unique athletes from the summer\_games table.
* Group by the non-aggregated field, which is sport.

1. **Age of oldest athlete by region**

Let's find out the age of the oldest athlete by region! We'll create a comprehensive report that includes the region and age\_of\_oldest\_athlete information. To achieve this, we'll combine data from three tables using two JOIN statements. By grouping the data based on the non-aggregated field, we'll gain valuable insights into the age distribution of athletes across different regions. Get ready to unravel fascinating findings about the oldest athletes! Let's get started by joining the tables and extracting the desired information.

* Create a report that shows region and age\_of\_oldest\_athlete.
* Include all three tables by running two separate JOIN statements.
* Group by the non-aggregated field.
* Alias each table AS the first letter in the table (in lower case).

1. **Number of events in each sport**

Counting the number of events in each sport for both summer and winter events is the goal here! To achieve this, we'll create a report that combines the unique events by sport from relevant tables using a UNION. Two GROUP BY statements will help organize the data as needed. Finally, we'll order the query to display the sports with the highest number of events first. Get ready to uncover the exciting world of sports events and their abundance across different disciplines! Let's proceed with the necessary steps to generate the desired report.

* Create a report that shows unique events by sport for both summer and winter events.
* Use a UNION to combine the relevant tables.
* Use two GROUP BY statements as needed.
* Order the final query to show the highest number of events first.

1. **Most decorated summer athletes**

Let's discover the most decorated summer athletes! By joining the summer\_games and athletes tables, we can select the athlete\_name and gold\_medals fields. We'll filter the table to only include athletes with at least 3 gold medals. Finally, we'll sort the table in descending order based on the number of gold medals to reveal the most accomplished athletes at the top. Get ready to unveil the names of the exceptional athletes who have achieved greatness in the summer games! Let's proceed with the necessary steps to obtain this prestigious list.

* Select athlete\_name and gold\_medals by joining summer\_games and athletes.
* Only include athlete\_name with at least 3 gold medals.
* Sort the table so that the most gold\_medals appears at the top.

1. **JOIN then UNION query**

Let's set up a powerful query to analyze unique events by country and season for summer events. We'll create a similar query for winter events. By combining the two queries using a UNION ALL, we can generate a comprehensive report that showcases events by country and season for both summer and winter. To make it even more informative, we'll sort the report in descending order based on the number of events. Get ready to uncover fascinating insights into the events held in different seasons and countries. Let's get started by setting up the queries and combining them using UNION ALL.

* Setup a query that shows unique events by country and season for **summer** events.
* Setup a similar query that shows unique events by country and season for **winter** events.
* Combine the two queries using a UNION ALL.
* Sort the report by events in descending order.

1. **CASE statement refresher**

Let's refresh our knowledge of the CASE statement for this exercise. Our goal is to create the "segment" field, which categorizes athletes into three segments: "Tall Female" for females who are at least 175 centimeters tall, "Tall Male" for males who are at least 190 centimeters tall, and "Other" for all other athletes. We'll use the height and gender fields from the athletes table and employ conditional logic (such as AND/OR) within CASE statements to accomplish this. Stay focused as we dive into building these segments and make use of the powerful CASE statement!

* In this exercise, your goal is to create the segment field that buckets an athlete into one of three segments:
* **Tall Female**, which represents a female that is at least 175 centimeters tall.
* **Tall Male**, which represents a male that is at least 190 centimeters tall.
* **Other**
* Each segment will need to reference the fields height and gender from the athletes table. Leverage CASE statements and conditional logic (such as AND/OR) to build this.
* Remember that each line of a case statement looks like this: CASE WHEN {condition} THEN {output}

1. **BMI bucket by sport**

Let's examine how BMI varies across different summer sports! To gain insights, we'll construct a report that includes the following information: sport (the name of the summer sport), bmi\_bucket (grouping BMI into three categories: <.25, .25-.30, >.30), and athletes (the count of unique athletes). We'll pull data from the summer\_games and athletes tables. Using a CASE statement, we'll split the bmi\_bucket without utilizing AND or ELSE conditions. Grouping the data by non-aggregated fields, we'll organize the report by sport and then athletes in descending order. Get ready to explore the relationship between BMI and summer sports! Let's proceed with building the query to generate the insightful report.

* You are looking to understand how BMI differs by each summer sport. To answer this, set up a report that contains the following:
* **sport**, which is the name of the summer sport
* **bmi\_bucket**, which splits up BMI into three groups: <.25, .25-.30, >.30
* **athletes**, or the unique number of athletes
* Definition: BMI = 100 \* weight / (height squared).
* Build a query that pulls from summer\_games and athletes to show sport, bmi\_bucket, and athletes.
* Without using AND or ELSE, set up a CASE statement that splits bmi\_bucket into three groups: '<.25', '.25-.30', and '>.30'.
* Group by the non-aggregated fields.
* Order the report by sport and then athletes in descending order.

1. **Filtering with a JOIN**

Let's create a concise report with the desired characteristics. The report will consist of three columns: bronze\_medals (total number of bronze medals), silver\_medals (total number of silver medals), and gold\_medals (total number of gold medals). We will only consider data from the summer\_games. Additionally, the report will be filtered to include only athletes aged 16 or under. We'll achieve this using the JOIN approach. Get ready to unveil the medal counts for young athletes in the summer games! Let's proceed with constructing the query to generate the insightful report.

* Your goal is to create a report with the following characteristics:
* First column is **bronze\_medals**, or the total number of bronze.
* Second column is **silver\_medals**, or the total number of silver.
* Third column is **gold\_medals**, or the total number of gold.
* Only summer\_games are included.
* Report is filtered to only include athletes age 16 or under.
* In this exercise, use the JOIN approach.

1. **FIltering with a subquery**

Let's tackle the task using a different approach this time. We'll create a subquery that outputs a list of athletes aged 16 or below. In the main query, we'll include a WHERE statement that references this subquery. The goal remains the same: create a report with three columns—bronze\_medals (total number of bronze medals), silver\_medals (total number of silver medals), and gold\_medals (total number of gold medals). We'll only consider data from the summer\_games and filter it to include athletes aged 16 or under. Get ready to generate the insightful report by setting up the query and subquery. Let's proceed with constructing it!

* Create a subquery that outputs a list.
* In your main query, add a WHERE statement that references the list.
* Your goal is to create the same report as the last exercise, which contains the following characteristics:
* First column is **bronze\_medals**, or the total number of bronze.
* Second column is **silve\_medals**, or the total number of silver.
* Third column is **gold\_medals**, or the total number of gold.
* Only summer\_games are included.
* Report is filtered to only include athletes age 16 or under.
* Create a query that pulls total bronze\_medals, silver\_medals, and gold\_medals from summer\_games.
* Setup a subquery that outputs all athletes age 16 or below.
* Add a WHERE statement that references the subquery to filter for athletes age 16 or below.
* In this exercise, use the subquery approach.

1. **Replacing and removing substrings**

Let's transform and filter the data in this task. We'll create two fields: character\_swap, which replaces '&' with 'and' in the region field, and character\_remove, which removes periods from the region field.

Next, we'll set up a query to retrieve the country and the total gold medals as gold\_medals for all winter games. The data will be grouped by the non-aggregated field and ordered in descending order based on gold\_medals.

Notice that null values appear at the top of the results. To remove these, we can either use a WHERE statement to filter out rows with null gold values or utilize a HAVING statement to exclude countries with no gold medals.

Let's proceed with executing the query, performing the necessary string manipulations, and applying the appropriate filtering to obtain the desired outcome.

* Create the field character\_swap
* that replaces all '&' characters with 'and' from region.
* Create the field character\_remove that removes all periods from region.Filtering out nulls
* Setup a query that pulls country and total golds as gold\_medals for all winter games.
* Group by the non-aggregated field and order by gold\_medals in descending order.
* Notice how null values appear at the top of the results. Remove these by adding a WHERE statement that filters out all rows with null gold values.
* We can do a similar filter using HAVING. Comment out the WHERE statement and add a HAVING statement that filters out countries with no gold medals.

1. **Fixing calculation with coalesce**

Let's perform some string manipulations and filtering in this task. We'll create two fields: character\_swap, which replaces all '&' characters with 'and' in the region field, and character\_remove, which removes all periods from the region field.

In the next query, we'll pull the country and total gold medals as gold\_medals for all winter games. We'll group the data by the non-aggregated field and order it in descending order based on gold\_medals.

Initially, we'll notice null values appearing at the top of the results. To remove these, we'll add a WHERE statement that filters out all rows with null gold values. Alternatively, we can achieve the same filtering using a HAVING statement. Simply comment out the WHERE statement and add a HAVING statement that filters out countries with no gold medals. Let's proceed with setting up the query to accomplish these tasks efficiently!

* Build a report that shows total\_events and gold\_medals by athlete\_id for all summer events, ordered by total\_events descending then athlete\_id ascending.
* Create a field called avg\_golds that averages the gold field.
* If the report was accurate, what should the first three values of avg\_golds be?
* A. [8, 8, 1]
* B. [0, 0, .125]
* C. [0, 0, 1]
* D. [0, 0, 8]
* Fix the avg\_golds field by replacing null values with zero.

1. **Testing out window functions**

Let's test out window functions to perform various calculations on GDP data. We'll start by adding the field country\_avg\_gdp, which outputs the average GDP for each country. In a separate code cell, we'll change country\_avg\_gdp to country\_sum\_gdp, displaying the total GDP for each country across all years. Further, we'll modify country\_sum\_gdp to country\_max\_gdp, showing the highest GDP for each country. Lastly, we'll transform country\_max\_gdp to global\_max\_gdp, revealing the highest GDP value for the entire world using the OVER() function. Get ready to explore different calculations using window functions! Let's proceed by implementing the required changes in separate code cells.

* Add the field country\_avg\_gdp that outputs the average gdp for each country.
* Change country\_avg\_gdp to country\_sum\_gdp that shows the total gdp for each country across all years in a different code cell.
* Change country\_sum\_gdp to country\_max\_gdp that shows the highest GDP for each country in a different code cell.
* Change country\_max\_gdp to global\_max\_gdp that shows the highest GDP value for the entire world in a different cell. Hint: Just use the OVER() with nothing inside the bracket.

1. **Average total country medals by region**

Let's analyze the average total country medals by region. We'll set up a query that retrieves the total gold medals by region and country\_id from the summer\_games\_clean and countries tables. We'll group the data by the non-aggregated fields. We'll then alias this query as a subquery and add another layer to calculate the average total gold medals for all countries in each region. Finally, we'll order the results by avg\_total\_golds in descending order. Get ready to uncover insights into the average medal count across regions! Let's proceed by setting up the query to generate the desired report.

* Set up a query that pulls total\_golds by region and country\_id from the summer\_games\_clean and countries tables.
* GROUP BY the unaggregated fields.
* Alias your query as subquery and add a layer that pulls region and avg\_total\_golds that outputs the average gold medal count for all countries in the region.
* Order by avg\_total\_golds in descending order.

1. **OPTIONAL**

* Create a report that makes use of RANK(), LEAD(), LAG(), ROLLUP(), and CUBE.