SQL CoP Problem Sheet 1 - Basic Query Statements

George Melrose: george_melrose@yahoo.com, https://github.com/georgemelrose

Introduction

These problems aim to test your basic SQL knowledge, steadily building up in complexity. The questions and solutions are common ones you will come across when querying different datasets. For the purposes of this series of problem sheets, a database of dummy Marathon results data has been generated. More information on the **Marathon** database is presented below.

The concepts tested in this sheet are covered by the LinkedIn learning course **SQL Server Fundamentals: Master Basic Query Techniques** - (https://www.linkedin.com/learning/sql-server-fundamentals-master-basic-query-techniques) .

Useful Prepatory Resources

In addition to this problem sheet, there are two useful resources you can draw upon to better understand these SQL concepts:

- **Two RMarkdown documents** - one to generate some dummy 'Universities' data (https://github.com/georgemelrose/SQL_Practice/blob/main/0_generating_databasestar_dummy_data.Rmd). This was copied from the excellent SQL learning resource databasestar (https://github.com/bbrumm/databasestar/tree/main/sample_databases/sample_db_university/sqlite).

  The other document is an RMD HTML I generated walking you through basic SQL concepts and how they can be applied to this databasestar dummy data (https://github.com/georgemelrose/SQL_Practice/blob/main/01_Basic_Query_Statements.html).

- **A video presentation** - a recording of a meeting in which I presented the **Basic Query Statements** HTML , explaining key SQL concepts - (https://universityofcambridgecloud.sharepoint.com/sites/AD_Progress/SitePages/Learning-SQL-in-a-New-Format.aspx). This video can be found on the aforementioned page under the **SQL and R** title. *Note that this can only be viewed by SQL" Community of Practice Members, email gam55@cam.ac.uk for access.*

Marathon Database

Firstly, the data to be put into the Marathon database was formulated from the following Python script - (https://github.com/georgemelrose/SQL_Practice/blob/main/Dummy_Marathon_Data/marathon_data_generation.ipynb).

The **marathon data generation** python script generates the following tables:

1. Runners - Randomly generate 1000 runners with names common in their locale/country, together with their birth date and sex.

2. Events - The 6 Major World marathons (Berlin, Boston, Chicago,London,New York City, Tokyo), with an event per year from 2012 to 2023.

3. Results - Gives results for runners in hh:mm:ss format, ensuring there aren't duplicate results for each runner per event. Prevents any results breaking either the male marathon world-record (2:00:35 Eliud Kipchoge 2023) or the female marathon world-record (2:11:53 Brigid Kosgei 2019). Also determines, with a True/False column, if a result is elite by the male standard (below 02:15:00) or the female standard (below 02:30:00).

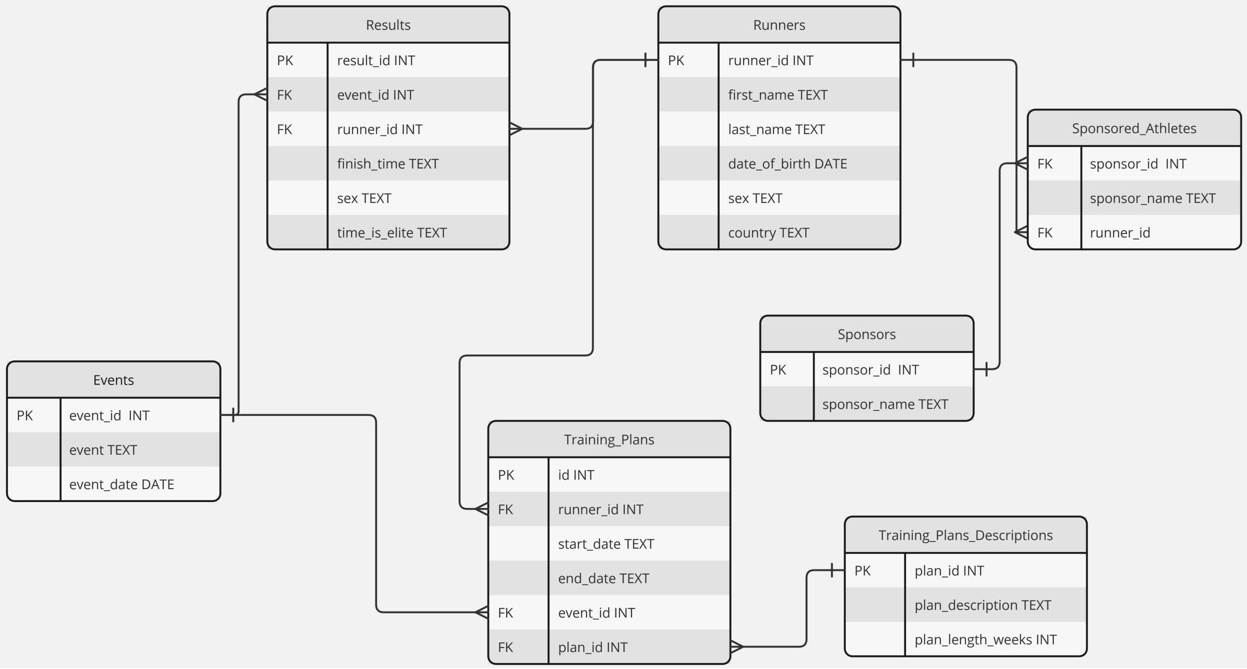
4. Sponsors - Lists the following 10 major companies that typically act as sponsors to runners - "Nike", "Adidas", "Asics", "Saucony", "Hoka","Brooks", "New Balance", "Puma", "Under Armour", "Tracksmith".

5. Sponsored Athletes - A table listing the fraction of the elite athletes that have a sponsor.

6. Training Plans Descriptions - The descriptions of 10 different training plans and their respective lengths in weeks.

7. Training Plans - The training plans of athletes. Only 72% of runner-event combinations have an associated training plan.

Marathon Database Entity Relationship Diagram



Results	
PK	result_id INT
FK	event_id INT
FK	runner_id INT
	finish_time TEXT
	sex TEXT
	time_is_elite TEXT

Runners	
PK	runner_id INT
	first_name TEXT
	last_name TEXT
	date_of_birth DATE
	sex TEXT
	country TEXT

Sponsored_Athletes	
FK	sponsor_id INT
	sponsor_name TEXT
FK	runner_id

Sponsors	
PK	sponsor_id INT
	sponsor_name TEXT

Events	
PK	event_id INT
	event TEXT
	event_date DATE

Training_Plans	
PK	id INT
FK	runner_id INT
	start_date TEXT
	end_date TEXT
FK	event_id INT
FK	plan_id INT

Training_Plans_Descriptions	
PK	plan_id INT
	plan_description TEXT
	plan_length_weeks INT

Basic Query Statment Problems

Single Table 'SELECT' Statements

Q1. - Get all the tables in the database, inspect their layout, and get the first 5 rows of each table?

Solutions -

.tables - to get all the tables.

.schema *table* - to get the schema of a particular table.

SELECT * FROM *table* limit 5 - to get the first 5 rows of a particular table.

Q2.i. - Obtain the full names and countries of the runners?

Solution -

```
SELECT first_name || ’ ’ || last_name AS full_name, country FROM runners;
```

The || symbols concatenate the first_name and last_name columns into one. The **AS** operator allows a column to have a different name displayed in the query result.

Q2.ii. - Find the top 10 Marathon results by time, male or female?

Solution -

```
SELECT * FROM Results ORDER BY finish_time ASC limit 10;
```

The **ORDER BY** operator orders the results by whichever column is specified. The **ASC** operator orders the results in ascending order, from the highest to the lowest time.

Filtering on Single Conditions

Q1 - Find the top 20 male marathon results and the top 20 female marathon results?

Solution -

```
SELECT * FROM Results WHERE sex= ’Male’ ORDER BY finish_time ASC limit 20;
SELECT * FROM Results WHERE sex= ’Female’ ORDER BY finish_time ASC limit 20;
```

The **WHERE** clause filters rows based on specified conditions, in the above cases the different sexes.

Q2.i. - Find all the runners from Lithuania, Latvia, and Estonia?

Solution -

```
SELECT * FROM Runners WHERE country in (’Lithuania’, ’Latvia’,’Estonia’);
```

Using the **in** operator, rows fulfilling a list can be found.

Q2.ii. - Find all the runners *not* from Poland, Czechia, Slovakia or Hungary?

Solution -

```
SELECT * FROM Runners WHERE country not in (’Poland’,’Czechia’,’Slovakia’,’Hungary’);
```

Using **not** in conjunction with the **in** operator, rows not fulfilling a list can be found.

Q3.i. - Find all the runners whose names begin with ‘J’?

Solution -

```
SELECT * FROM Runners WHERE first_name like ’J%’;
```

Using **WHERE** clause in conjunction with the **like** operator, rows with a first name beginning with J can be found. Note that in string matching, ‘string%’ find a pattern at the beginning of a string. ‘%string’ means finding a pattern at the end of a string.

Q3.ii. - Find all the runners whose names don’t begin with ‘J’?

Solution -

```
SELECT * FROM Runners WHERE first_name not like ’J%’;
```

The answer is exactly the same as in part i. albeit with the **not** operator to signify everything *not* fulfilling the condition.

Q3.iii. - Find all the runners whose surnames contain ‘son’?

Solution -

```
SELECT * FROM Runners WHERE last_name like ’%son%’;
```

In this solution, all the matching rows that *contain* the son string regardless WHERE it is in the names are found by wrapping the string in % signs in the query code.

Q4.i. - Gather all training plans that have a length between 10 and 12 weeks?

Solution -

SELECT * FROM Training_Plans_Descriptions WHERE plan_length_weeks between 10 and 12;

The **between** operator together with the required numerical parameters and the **and** operator

Q4.ii. - Gather all training plans that have a length less than 12 weeks?

Solution -

SELECT * FROM Training_Plans_Descriptions WHERE plan_length_weeks <12;

The less than < operator after the **WHERE** clause fetches results below a certain numerical threshold, 12 in this case.

Q4.iii. - Gather all training plans that have a length more than 12 weeks?

Solution -

SELECT * FROM Training_Plans_Descriptions WHERE plan_length_weeks >12;

The greater than > operator after the **WHERE** clause fetches results above a certain numerical threshold, 12 in this case.

Q4.iv. - Gather all training plans that have a length more than or equal to 12 weeks?

Solution -

SELECT * FROM Training_Plans_Descriptions WHERE plan_length_weeks >=12;

The greater than or equals to >= operator after the **WHERE** clause fetches results above or equal to a certain numerical threshold, 12 in this case.

Filtering on Multiple Conditions

Q1.i. - Find all female runners from the United Kingdom born in the 20th century?

Solution -

SELECT * FROM Runners WHERE sex = 'Female' and country = 'United Kingdom' and date_of_birth > 01-01-2000;

The greater than > operator is used, converse to normal logic, to find dates *less than* 1st January 2000.

Q1.ii. - Find all male runners from Brazil born in 2000?

Solution -

SELECT * FROM Runners WHERE sex = 'Male' and country = 'Brazil' and date_of_birth between '2000-01-01' and '2000-12-31';

In this solution, despite the date column having the format yyyy-mm-dd we can still whittle the results down to the year. Using the **between** and **and** operators, together with the date range we seek, those rows fulfilling the question's conditions can be found.

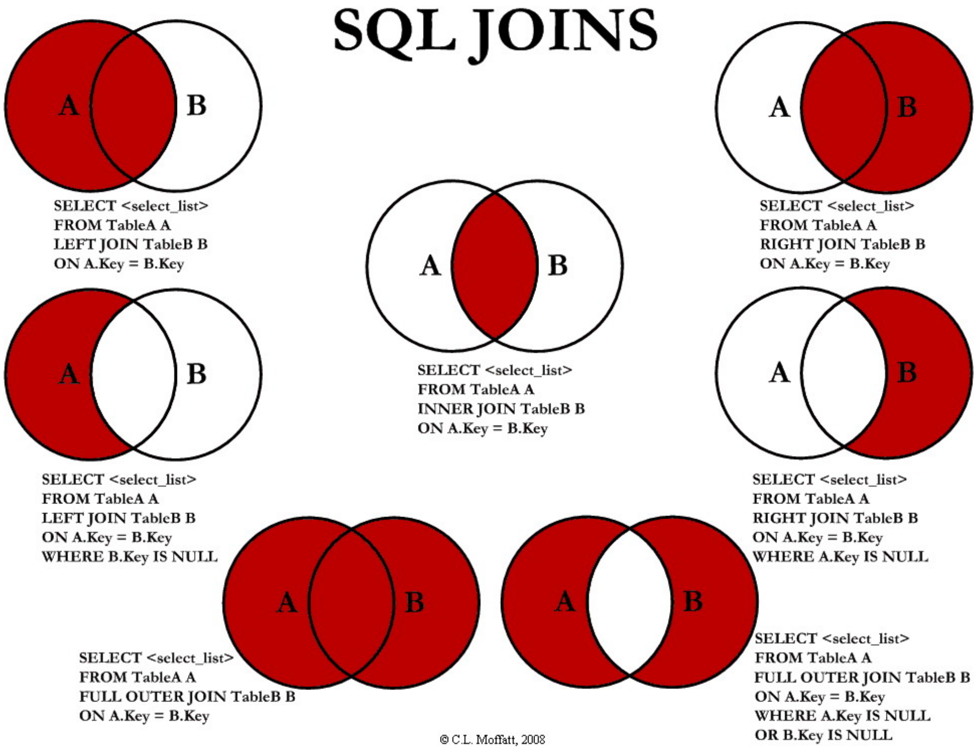
Q2. - Find the full names of all runners whose names begin with the letters G or J?

Solution -

SELECT first_name || ' ' || last_name AS full_name FROM Runners WHERE first_name like 'J%' or first_name like 'G%';

To get the full names of runners the first_name and last_name columns are combined using the || for string concatenation, with a new column in the query result - *full_name*.As in the ‘Filtering on Single Conditions’ section, string matching and the **WHERE** operator are used to obtain rows fulfilling these multiple conditions.

Single Inner Joins



```
SELECT <select_list>
FROM TableA A
LEFT JOIN TableB B
ON A.Key = B.Key
```

```
SELECT <select_list>
FROM TableA A
INNER JOIN TableB B
ON A.Key = B.Key
```

```
SELECT <select_list>
FROM TableA A
RIGHT JOIN TableB B
ON A.Key = B.Key
```

```
SELECT <select_list>
FROM TableA A
LEFT JOIN TableB B
ON A.Key = B.Key
WHERE B.Key IS NULL
```

```
SELECT <select_list>
FROM TableA A
RIGHT JOIN TableB B
ON A.Key = B.Key
WHERE A.Key IS NULL
```

```
SELECT <select_list>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.Key = B.Key
```

```
SELECT <select_list>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.Key = B.Key
WHERE A.Key IS NULL
OR B.Key IS NULL
```

© C.L. Moffatt, 2008

Q1.i. - Obtain the runner ids of runners with an elite time in the London Marathon(any year)?

Solution -

SELECT runner_id FROM Results inner join Events on Results.event_id = Events.event_id WHERE time_is_elite = ’True’ and event = ’London Marathon’;

A single inner join with the Results table allows for a filter on the condition of the event being the London marathon as well as on the condition of ‘time_is_elite’ to be made.

Q1.ii. - Obtain the number of runner ids of runners with an elite time in the London Marathon(any year)?

Solution -

SELECT count(runner_id) FROM Results inner join Events on Results.event_id = Events.event_id WHERE time_is_elite = ’True’ and event = ’London Marathon’;

Using the **count()** function, the number of runner ids in the query result are counted.

Q1.iii. - Obtain the number of unique runner ids of runners with an elite time in the London Marathon(any year)?

Solution -

```
SELECT count(distinct runner_id) FROM Results inner join Events on Results.event_id = Events.event_id WHERE time_is_elite = ’True’ and event = ’London Marathon’;
```

Using **distinct** in conjunction with the **count()** function, the number of unique runner ids in the query result are counted.

Q2. - Find the the following information for male runners of any of the Boston Marathon events in the database: runner_id; finish_time; time_is_elite?

Solution -

```
SELECT runner_id, finish_time,sex, time_is_elite FROM Results inner join Events on Events.event_id = Results.event_id WHERE event = ’Boston Marathon’ and sex = ’Male’;
```

The 4 necessary columns are selected from the Results table, which is then joined on to the Events table by the column they have in common - event_id. Filtering specifically down to the Boston Marathon events is done by using a **WHERE** clause.

Multiple Inner Joins

Q1.i. - Find the full names of runners from the UK who had an elite time in the London Marathon (any year)?

Solution -

```
SELECT first_name || ’ ’ || last_name AS full_name FROM Runners inner join Results on Results.runner_id = Runners.runner_id inner join Events on Events.event_id = Results.event_id WHERE event = ’London Marathon’ and time_is_elite = ’True’ and country = ’United Kingdom’;
```

To obtain the full names of runners the first_name and last_name columns are combined using the || for string concatenation, with a new column in the query result - *full_name.*Two inner joins are made. The first joining the Results table on to the Runners table using the common variable, runner_id. The second joining the Events table with the Results table using the variable they have in common, event_id. WHERE clauses for the columns ‘event’, ‘time_is_elite’ and ‘country’ ensure the query result is filtered accordingly.

Q1.ii. - Find the number of male and female runners respectively, from the UK, who had an elite time in the London Marathon (any year)?

Solution -

```
SELECT first_name || ’ ’ || last_name AS full_name FROM Runners inner join Results on Results.runner_id = Runners.runner_id inner join Events on Events.event_id = Results.event_id WHERE event = ’London Marathon’ and time_is_elite = ’True’ and country = ’United Kingdom’ and Runners.sex = ’Male’;
```

```
SELECT first_name || ’ ’ || last_name AS full_name FROM Runners inner join Results on Results.runner_id = Runners.runner_id inner join Events on Events.event_id = Results.event_id WHERE event = ’London Marathon’ and time_is_elite = ’True’ and country = ’United Kingdom’ and Runners.sex = ’Female’;
```

Adding the WHERE clause for ‘sex’ is obtains the answer. Additionally, the table name needs to be specified as the ‘sex’ column is present in both the Runners and Results tables.

Q2. - Find the full names, finish time and countries of sponsored runners from the UK who had an elite time in the Berlin Marathon (any year)?

Solution -

```
SELECT r.first_name || ’ ’ || r.last_name AS full_name,r.country, sa.sponsor_name, re.finish_time FROM Runners r inner join Sponsored_Athletes sa on sa.runner_id =
```

r.runner_id inner join Results re on re.runner_id = r.runner_id inner join Events e on e.event_id = re.event_id WHERE event = ’Berlin Marathon’ and time_is_elite = ’True’;

As different columns from different tables need to be selected, aliases are essential. Each table is given an alias which then ensure the selection of columns is clear. Inner joins to the different tables aside FROM our starting point of Runners are made - joins to Sponsored_Athletes sa, Results re, and Events e.

Left Outer Joins

Q1. - Find the full names,sex, and country of runners sponsored by Nike?

Solution -

SELECT r.first_name || ’ ’ || r.last_name AS full_name,r.country, r.sex FROM Runners r LEFT OUTER JOIN Sponsored_Athletes sa on r.runner_id = sa.runner_id WHERE sponsor = ’Nike’;

Using aliases is helpful again, with several tables present in the query. The **LEFT OUTER JOIN** function gets all relevant results from the ‘left’ table, Runners, that matches with the key of the ‘right’ table Sponsored_Athletes.

Q2.i. - Find dates and times of results that had a finish time under 02:10:00 in Chicago?

Solution -

SELECT e.event_date, r.finish_time from Events e LEFT OUTER JOIN Results r on e.event_id = r.event_id WHERE STRFTIME(’%H:%M:%S’,r.finish_time) < ’02:10:00’ and e.event = ’Chicago Marathon’;

Aliases allow the smooth selection of columns from multiple tables. As the Results finish_time column is in text form, the **STRFTIME()** function needs to be utilised to convert it into a comparable format.

Q2.ii. - Count the number of distinct results that had a finish time under 02:10:00 in Chicago?

Solution -

SELECT COUNT(DISTINCT r.finish_time) from Events e LEFT OUTER JOIN Results r on e.event_id = r.event_id WHERE STRFTIME(’%H:%M:%S’,r.finish_time) < 02:10:00 and e.event = ’Chicago Marathon’;

As the question does not specify any dates being in the result, they are not selected. COUNT() and DISTINCT are used to obtain the number of distinct results.

Subqueries

Q1. - Using a subquery, find the full names, birth dates, and countries of runners who achieved a time below 02:20:00?

Solution -

SELECT r.first_name || ’ ’ || r.last_name AS full_name, r.country, r.sex FROM Runners r WHERE r.runner_id IN (SELECT re.runner_id FROM Results re WHERE STRFTIME(’%H:%M:%S’, re.finish_time) < ’02:20:00’);

The solution is as follows: aliases of r for the Runners and re for the Results tables respectively to be able to differentiate between columns; concatenate the first_name and last_name columns using the || concatenation operator; insert a subquery by using two **WHERE** statements, the **STRFTIME()** function and < operator to get runner IDs that have a time below 2 hours and 20 minutes.

Q2. - Using a subquery, find all the runners by full name and sex who are sponsored by Nike or Tracksmith?

Solution -

```
SELECT r.first_name || ’ ’ || r.last_name AS full_name, r.country, r.sex FROM Runners r
WHERE r.runner_id IN ( SELECT sa.runner_id FROM Sponsored_Athletes sa WHERE
sa.sponsor_name = ’Nike’ OR sa.sponsor_name = ’Tracksmith’);
```

As with question 1 the || concatenation operator is utilised to obtain the full name in the query result. Aliases are used to distinguish successfully between the two tables. A subquery allows the filtering of runners to just those who with Nike or Tracksmith as a sponsor, through repeated use of the **WHERE** operator on the Sponsored_Athletes table.

Q3. - Using a subquery, find all the male runners ids and their times, when their times were elite in the 2023 Boston Marathon?

Solution -

```
SELECT r.runner_id, r.finish_time FROM Results r WHERE r.sex = ’Male’ AND
time_is_elite = ’True’ r.event_id IN ( SELECT sa.runner_id FROM Sponsored_Athletes
sa WHERE sa.sponsor_name = ’Nike’ OR sa.sponsor_name = ’Tracksmith’);
```

Case When Statements

Q1. - Using a subquery and a case when statement, categorise of the 2023 London Marathon in the following way: >3 hours; 02:30:00 to 3hours; below 02:30:00 ? Provide the runner’s full name, date_of_bith, sex, and country in the query result? Order from fastest to slowest?

Solution -

```
SELECT r.first_name || ’ ’ || r.last_name AS full_name, r.country, r.sex, re.finish_time,
CASE WHEN STRFTIME(’%H:%M:%S’, re.finish_time) > ’03:00:00’ THEN ’>3 Hours’
WHEN STRFTIME(’%H:%M:%S’, re.finish_time) BETWEEN ’02:30:00’ AND ’03:00:00’
THEN ’2.5 to 3 Hours’ WHEN STRFTIME(’%H:%M:%S’, re.finish_time) < ’02:30:00’
THEN ’<2.5 Hours’ ELSE ’NA’ END AS Top_Runners FROM Runners r INNER JOIN
Results re ON re.runner_id = r.runner_id WHERE re.event_id IN (SELECT e.event_id
FROM Events e WHERE event = ’London Marathon’ AND event_date = ’2023-04-23’)
ORDER BY re.finish_time ASC;
```

As in previous answers, the || concatenation operator is utilised to obtain a runner’s full name. Aliases are used to distinguish successfully between the two tables.

In the **CASE WHEN** statement, the **STRFTIME()** function is employed to put the text column of finish_time from the Results table into a comparable time format. The Runners and Results tables are joined to one another using **INNER JOIN**, on the column in common runner_id.

In the penultimate part of the query, a subquery is present. The subquery ensures that the only rows in the Results table included in the query result, are those with event = ‘London Marathon’ AND event_date = ‘2023-04-23’. Finally, the query result is ordered by the the finishing times from fastest to slowest (the **ASC** operator).

Q2. - Utilising a case when statement, categorise all runners by which country their Sponsor is from (Adidas & Puma - Germany) (Asics - Japan) (Nike, Saucony, Hoka, Brooks, New Balance,Under Armour, Tracksmith), providing their full name, date of birth, sex, and country in the query result?

Solution -

```
SELECT r.first_name || ’ ’ || r.last_name AS full_name, r.country, r.sex, r.date_of_birth,
sa.sponsor_name, CASE WHEN sa.sponsor_name IN (’Adidas’, ’Puma’) THEN ’Germany’
WHEN sa.sponsor_name = ’Asics’ THEN ’Japan’ WHEN sa.sponsor_name NOT IN (’Adidas’,’Puma’,’Asics’) THEN ’USA’ ELSE ’NA’ END AS Sponsor_Country FROM Runners r
INNER JOIN Sponsored_Athletes sa on sa.runner_id = r.runner_id;
```

In the solution query, the || concatenation operator is utilised to obtain a runner’s full name. Aliases are used to distinguish successfully between the two tables. Lists are used in the CASE WHEN statements to assign different sponsors to their respective countries.

The two operators **NOT** and **IN** are used instead of making a list of the many sponsors that are American. An INNER JOIN joins sponsored_athletes and runners together on the runner ids.

Q3. - Using a case when statement, generate the following categories for the different generations present amongst the runners: Silent Generation; Boomers; Gen X, Millennials; Gen Z? Include their full name, country, sex, and date of birth?

Solution -

```
SELECT first_name || ’ ’ || last_name AS full_name, country, sex, date_of_birth,
CASE WHEN date_of_birth between ’1928-01-01’ AND ’1945-12-31’ THEN ’Silent
Generation’ WHEN date_of_birth between ’1946-01-01’ AND ’1964-12-31’ THEN
’Boomer’ WHEN date_of_birth between ’1965-01-01’ AND ’1980-12-31’ THEN ’Gen.
X’ WHEN date_of_birth between ’1981-01-01’ AND ’1996-12-31’ THEN ’Millennial’ WHEN
date_of_birth between ’1997-01-01’ AND ’2012-12-31’ THEN ’Gen. Z’ ELSE ’Unknown’
END AS Generation FROM Runners;
```

This solution above does not have any aliases as only 1 table is being used. The CASE WHEN statement consists of several WHENs, with the **BETWEEN** operator allowing a date ranges to be assigned to corresponding generations.

Aggregate Functions

Q1. - Count how many rows there are in each table, using an aggregate function?

Solution -

```
SELECT COUNT(*) FROM TABLE_NAME
```

The **COUNT()** function performs calculations across multiple rows to find a sum or an average.

Q2. - Count how many runners are present per country in the Runners table?

Solution -

```
SELECT COUNT(*) AS "No.Runners", country FROM Runners GROUP BY country;
```

The **COUNT(*)** function is used alongside **GROUP BY** to achieve the desired result of number of runners per country.

Q3. - Count how many runners obtained an elite time in any Berlin Marathon event?

Solution -

```
SELECT COUNT(DISTINCT runner_id) FROM Results WHERE time_is_elite = ’True’
AND event_id IN (SELECT event_id FROM Events WHERE event = ’Berlin Marathon’);
```

Using **COUNT()** and **DISTINCT()** together as well as a subquery, the number of runners obtaining an elite time in the Berlin Marathon can be found - 840.

Query Processing Order

A SQL query is processed in the following order (not necessarily written in this order) -

1. **FROM Clause** - The query starts by identifying the tables from which data will be retrieved. If there are any JOIN operations, they are also processed at this stage to create the base dataset.

2. **WHERE Clause -** After forming the initial dataset from the FROM clause, the WHERE clause is applied to filter rows based on specified conditions.

3. **GROUP BY Clause -** The data is then grouped according to the columns specified in the GROUP BY clause.

4. **HAVING Clause -** The HAVING clause is applied next, which allows filtering groups created by the GROUP BY clause based on aggregate conditions.

5. **SELECT Clause -** The SELECT clause is then processed to determine which columns or expressions should be included in the final result.

6. **ORDER BY Clause -** The ORDER BY clause sorts the result set based on the specified columns or expressions.

7. **LIMIT / OFFSET Clause -** If present, the LIMIT and OFFSET clauses are applied to restrict the number of rows returned by the query, and to skip a specified number of rows, respectively.

Q1. - Bearing the above in mind, generate a query that the top 10 fastest times of athletes at the 2016 London Marathon, with their full names, dates of birth, sexes, countries, sponsors, and training plans included in the query result?

Solution -

```
SELECT re.finish_time, r.first_name || ’ ’ || r.last_name AS full_name, r.date_of_birth,r.sex, r.country, tpd.plan_description FROM Runners r INNER JOIN Results re ON re.runner_id = r.runner_id INNER JOIN Training_Plans tp ON tp.runner_id = r.runner_id AND tp.event_id = re.event_id INNER JOIN Training_Plans_Descriptions tpd ON tpd.plan_id = tp.plan_id INNER JOIN Events e ON e.event_id = re.event_id WHERE e.event = ’London Marathon’ AND e.event_date = ’2016-04-24’ ORDER BY re.finish_time ASC LIMIT 10;
```

Firstly, usage of aliases is crucial for a succinct and readable query in this case as 5 different tables are being read. The first_name and last_name columns from Runners are concatenated again as they have been in multiple queries in other sections.

Four INNER JOINs are done to join Runners to Results, Results to Training_Plans, Training_Plans to Training_Plans_Descriptions, and Events to Results. Specifically, the INNER JOIN on ‘Training_Plans’ is on *two* variables - runner_id and event_id.

This is to ensure that the Training_Plans entry corresponds to the specific event the runner participated in (the 2016 London Marathon), avoiding any accidental mismatches with other events.

The Training_Plans table has a unique association of runners and events, so joining on both runner_id and event_id ensures the correct training plan is retrieved for that particular event.

A WHERE clause towards the end of the query ensures that the scope of the query result is only the 2016 London Marathon. The ORDER BY and ASC operators order the results from fastest to slowest and then LIMIT 10 limits the query result to 10 rows.

Q1. - Bearing the above in mind, generate a query that the top 10 fastest times of athletes at the 2016 London Marathon, with their full names, dates of birth, sexes, countries, sponsors, and training plans included in the query result?