This is a supplementary explanation of query statements in QLab *Working with JSON, Arrays, and Structs in BigQuery* focusing on Arrays and Structs. We take table 'racing.race_results' as an example.

Here's part of the table and the schema:

Row	race	participants.name	participants.splits
1	800M	Rudisha	23.4
			26.3
			26.4
			26.1
		Makhloufi	24.5
			25.4
			26.6
			26.1
		Murphy	23.9
			26
			27
			26
		...	...

Schema Details Preview

Field name	Type	Mode
race	STRING	NULLABLE
participants	RECORD	REPEATED
participants. name	STRING	NULLABLE
participants. splits	FLOAT	REPEATED

1. Let's first look at the schema. We know **participants** is a Struct as well as an Array, **participants.name** and **participants.splits** are elements of the Struct **participants**, and **participants.splits** is an Array. We start with querying each column out of the table.

1) For column **race**, we could directly query it from the table as what we do before - it's neither a Struct nor an Array. So the query statement should be as follows:

```
SELECT race
FROM racing.race_results
```

2) For column **participants.name**, the following query will raise error:

```
SELECT participants.name
FROM racing.race_results
```

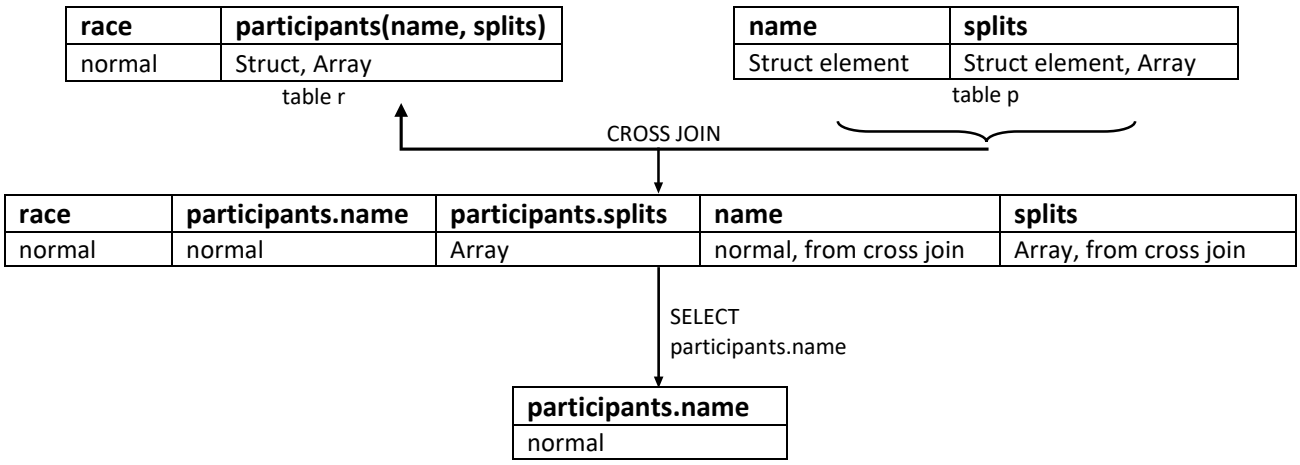
This is because we didn't unpack **participants** before accessing it. Keep in mind 'STRUCTs (and ARRAYs) must be unpacked before you can operate over their elements.' We could revise the statement in either way:

```
SELECT p.name
FROM racing.race_results AS r, r.participants AS p
```

or

```
SELECT p.name
FROM racing.race_results, UNNEST(participants) AS p
```

Both two queries are valid and the results are the same. The comma here indicates an implicit CORSS JOIN. That is, cross join table r with table p on **participants**:



Notice that we could either SELECT p.name or SELECT name, because after cross join we have two columns named **participants.name** and **name** respectively. The results are the same. Likewise, if we want to select only race time, then SELECT p.splits and SELECT splits both work.

Of course you can write the query using explicit join. That is,

```
SELECT p.name
FROM racing.race_results AS r
CROSS JOIN r.participants AS p
```

Then what about the following one?

```
SELECT name
FROM racing.race_results.participants
```

You will get an error: 'The project racing has not enabled BigQuery.' This is because BigQuery automatically detects a table in absolute path '<project-name>.<dataset-name>.<table-name>'. In this case, 'racing.race_results.participants' is mistakenly recognized as a table from project 'racing', dataset 'race_results', which makes no sense.

Also, as mentioned before, we cannot operate over elements within an Array or Struct without unpacking it. 'name' cannot be accessed without specifying and unpacking the Struct that contains it.

3) Similar to **participants.name**, we could retrieve **participants.splits** by:

```
SELECT p.splits
FROM racing.race_results AS r, r.participants AS p
```

or

```
SELECT p.splits
FROM racing.race_results, UNNEST(participants) AS p
```

4) What if we want to get all elements in **participants**? Try the following one:

```
SELECT participants.*
FROM racing.race_results
```

This query statement is invalid. Compared it with the one in section 'Introduction to STRUCTs'; the problem here is that **participants** is not only a Struct, but an Array as well, whereas 'totals' and 'device' in table 'bigquery-public-data.google_analytics_sample.ga_sessions_20170801' are just Structs. Dot-star is not supported for Array type variables.

Revise it using UNNEST() function. This time the query works well.

```
SELECT p.*
FROM racing.race_results, UNNEST(participants) AS p
```

Similarly, we can also revise it in this way:

```
SELECT p.*
FROM racing.race_results AS r, r.participants AS p
```

2. Now we could move further to more complicated queries.

1) How can we count the number of racers in total? (Lab Question: STRUCT())

We could begin with the query 2) above. Add an aggregation function COUNT() to it:

```
SELECT COUNT(p.name) AS racer_count
FROM racing.race_results AS r, r.participants AS p
```

or

```
SELECT COUNT(p.name) AS racer_count
FROM racing.race_results, UNNEST(participants) AS p
```

Both two queries are valid and generate the same results.

2) How can we find out the best record and the racer who created it? We now need to refer to an element of a Struct in an ORDER BY clause. We may try this one:

```
SELECT p.name, p.splits AS split_time
FROM racing.race_results, UNNEST(participants) AS p
ORDER BY p.splits
LIMIT 1
```

Again, we didn't unpack Array **participants.splits** before accessing it, so this statement is invalid. We need an extra UNNEST() function to unpack **participants.splits**. A valid one could be like:

```
SELECT p.name, split_time
FROM racing.race_results, UNNEST(participants) AS p, UNNEST(p.splits) AS split_time
ORDER BY split_time
LIMIT 1
```

Of course we could replace the FROM clause in the previous query with:

```
FROM racing.race_results AS r, r.participants AS p, UNNEST(p.splits) AS split_time
```

The results are exactly the same.

3) How can we find out the total race time for racers whose names begin with R, and order the results with the fastest total time first? (Lab Question: Unpacking ARRAYs with UNNEST( ))

We can decompose the problem into three parts:

(i) SUM the race time for every racer;

(ii) Match racer names using LIKE and wildcards;

(iii) ORDER BY time in ascending order.

We may start from the previous query statement and add a SUM() function for variable **split_time**. Let's try:

```
SELECT
   p.name,
   SUM(split_time) AS total_race_time
FROM
   racing.race_results, UNNEST(participants) AS p, UNNEST(p.splits) AS split_time
GROUP BY p.name
```

Step (i) done. Don't forget to add a GROUP BY clause since we use aggregation function SUM() in the SELECT clause.

Now let's add a filtering condition for racer's name using WHERE clause:

```
SELECT
   p.name,
   SUM(split_time) AS total_race_time
FROM
   racing.race_results, UNNEST(participants) AS p, UNNEST(p.splits) AS split_time
WHERE p.name LIKE 'R%'
```

```
GROUP BY p.name
```

Finally, add an ORDER BY clause to sort the result:

```
SELECT
   p.name,
   SUM(split_time) AS total_race_time
FROM
   racing.race_results, UNNEST(participants) AS p, UNNEST(p.splits) AS split_time
WHERE p.name LIKE 'R%'
GROUP BY p.name
ORDER BY total_race_time
```

Done. Again, we could replace the FROM clause above with:

```
FROM
   racing.race_results AS r, r.participants AS p, UNNEST(p.splits) AS split_time
```