

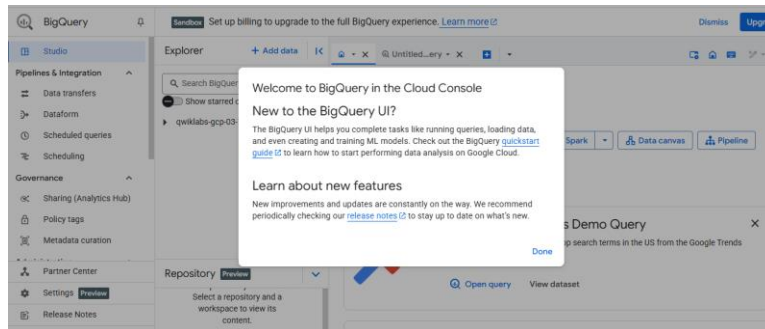
## Working with JSON, Arrays, and Structs in BigQuery

Open the BigQuery console

1. In the Google Cloud Console, select **Navigation menu > BigQuery**.

The **Welcome to BigQuery in the Cloud Console** message box opens. This message box provides a link to the quickstart guide and the release notes.

2. Click **Done**.



### Task 1. Create a new dataset to store the tables

1. In your BigQuery, click the three dots next to your Project ID and select **Create dataset**:
2. Set the **Dataset ID** to fruit\_store. Leave the other options at their default values (Data Location, Default Expiration).
3. Click **Create dataset**.

Create dataset

Project ID \*  
qwiklabs-gcp-03-7072cb4c98d9 [Change](#)

Dataset ID \*  
fruit\_store  
Letters, numbers, and underscores allowed

Location type [?](#)

☐ Region  
Specify a region to colocate your datasets with other Google Cloud services.

☒ Multi-region  
Allow BigQuery to select a region within a group to achieve higher quota limits.

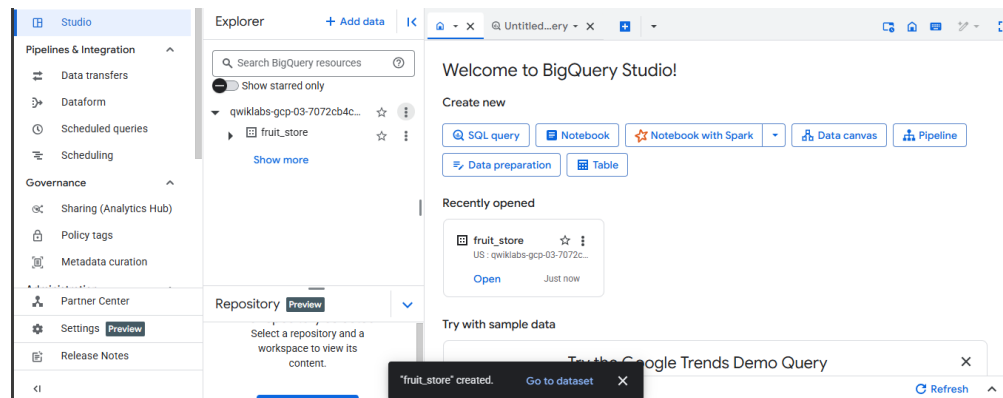
**i** Some locations have been restricted due to a policy set by your organization. [Learn more about restricting locations.](#)

Multi-region \*  
US (multiple regions in United States)

External Dataset  
The selected region supports the following external dataset types: Cloud Spanner

☐ Link to an external dataset [?](#)

[Create dataset](#) [Cancel](#)



## Task 2. Practice working with arrays in SQL

Normally in SQL you will have a single value for each row like this list of fruits below:

Row	Fruit
1	raspberry
2	blackberry
3	strawberry

4	cherry
---	--------

What if you wanted a list of fruit items for each person at the store? It could look something like this:

Row	Fruit	Person
1	raspberry	sally
2	blackberry	sally
3	strawberry	sally
4	cherry	sally
5	orange	frederick
6	apple	frederick

In traditional relational database SQL, you would look at the repetition of names and immediately think of splitting the above table into two separate tables: Fruit Items and People. That process is called [normalization](#) (going from one table to many). This is a common approach for transactional databases like MySQL.

For data warehousing, data analysts often go the reverse direction (denormalization) and bring many separate tables into one large reporting table.

Now, you're going to learn a different approach that stores data at different levels of granularity all in one table using repeated fields:

Row	Fruit (array)	Person
1	raspberry	sally
	blackberry	

	strawberry	
	cherry	
	orange	
	apple	
2		frederick

What looks strange about the previous table?

- It's only two rows.
- There are multiple field values for Fruit in a single row.
- The people are associated with all of the field values.

What the key insight? The array data type!

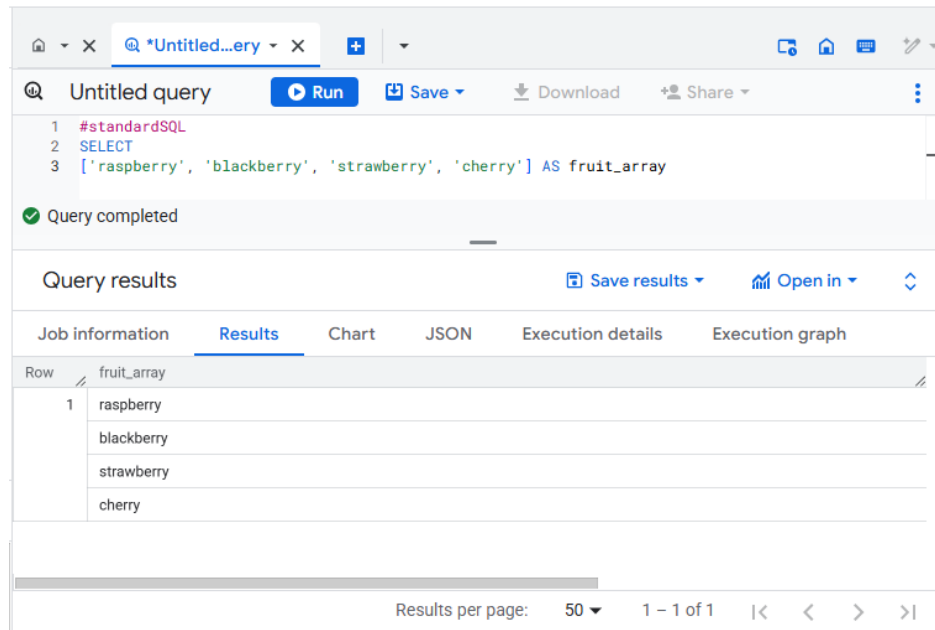
An easier way to interpret the Fruit array:

Row	Fruit (array)	Person
1	[raspberry, blackberry, strawberry, cherry]	sally
2	[orange, apple]	frederick

Both of these tables are exactly the same. There are two key learnings here:

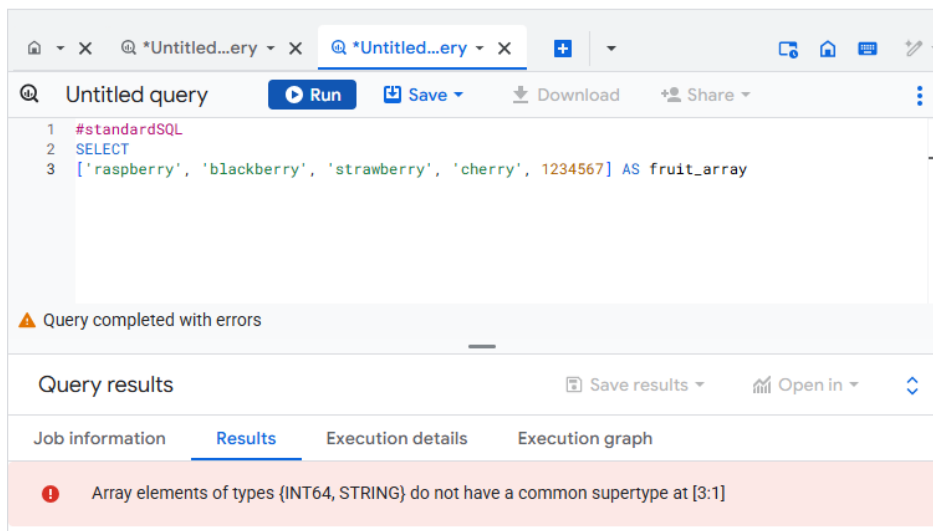
- An array is simply a list of items in brackets [ ]
- BigQuery visually displays arrays as *flattened*. It simply lists the value in the array vertically (note that all of those values still belong to a single row)

1. Enter the following in the BigQuery Query Editor:



2. Click **Run**.

3. Now try executing this one:



You should get an error that looks like the following:

**Error:** Array elements of types {INT64, STRING} do not have a common supertype at [3:1]

4. Here's the final table to query against:

The screenshot shows the BigQuery console interface. At the top, there's a query editor with two lines of SQL: `#standardSQL` and `SELECT person, fruit_array, total_cost FROM `data-to-insights.advanced.fruit_store`;`. Below the editor, a green checkmark indicates 'Query completed'. The 'Query results' section is active, displaying a table with three columns: 'person', 'fruit\_array', and 'total\_cost'. The table has two rows of data. The first row is for 'sally' with fruit\_array containing 'raspberry', 'blackberry', 'strawberry', and 'cherry', and a total\_cost of 10.99. The second row is for 'frederick' with fruit\_array containing 'orange' and 'apple', and a total\_cost of 5.55. The interface includes tabs for 'Job information', 'Results', 'Chart', 'JSON', 'Execution details', and 'Execution graph'. At the bottom, it shows 'Results per page: 50' and '1 - 2 of 2'.

Row	person	fruit_array	total_cost
1	sally	raspberry blackberry strawberry cherry	10.99
2	frederick	orange apple	5.55

5. Click **Run**.

6. After viewing the results, click the **JSON** tab to view the nested structure of the results.

This screenshot shows the same BigQuery console interface, but the 'JSON' tab is selected under 'Query results'. It displays the query results in a nested JSON structure. The JSON array contains two objects, one for 'sally' and one for 'frederick', each with their respective fruit arrays and total costs.

```
[{
  "person": ["sally"],
  "fruit_array": ["raspberry", "blackberry", "strawberry", "cherry"],
  "total_cost": ["10.99"]
}, {
  "person": ["frederick"],
  "fruit_array": ["orange", "apple"],
  "total_cost": ["5.55"]
}]
```

## Loading semi-structured JSON into BigQuery

What if you had a JSON file that you needed to ingest into BigQuery?

Create a new table `fruit_details` in the dataset.

1. Click on `fruit_store` dataset.

Now you will see the **Create Table** option.

2. Add the following details for the table:
  - **Source:** Choose **Google Cloud Storage** in the **Create table from** dropdown.
  - **Select file from Cloud Storage bucket:** cloud-training/data-insights-course/labs/optimizing-for-performance/shopping\_cart.json
  - **File format:** JSONL (Newline delimited JSON)
3. Call the new table fruit\_details.
4. Check the checkbox of **Schema (Auto detect)**.
5. Click **Create table**.

Create table

Source

Create table from  
Google Cloud Storage

Select file from GCS bucket or use a URI pattern \*  
☒ cloud-training/data-insights-course/labs/optimizing-for-performance/shopping\_cart.json Browse ⓘ

File format  
JSONL (Newline delimited JSON)

☐ Source Data Partitioning

Destination

Project \*  
qwiklabs-gcp-03-7072cb4c98d9 Browse

Dataset \*  
fruit\_store

Table \*  
fruit\_details

Maximum name size is 1,024 UTF-8 bytes. Unicode letters, marks, numbers, connectors, dashes, and spaces are allowed.

Table type

Create table Cancel

### Task 3. Create your own arrays with ARRAY\_AGG()

Don't have arrays in your tables already? You can create them!

1. **Copy and paste** the below query to explore this public dataset:

Untitled query Run Save Download Share

```

1 SELECT
2   fullVisitorId,
3   date,
4   v2ProductName,
5   pageTitle
6 FROM `data-to-insights.ecommerce.all_sessions`
7 WHERE visitId = 1501570398
8 ORDER BY date

```

Query completed

Query results Save results Open in

Job information	Results	Chart	JSON	Execution details	Execution graph
Row	fullVisitorId	date	v2ProductName	pageTitle	
1	5710379250208908569	20170731	Google Snapback Hat Black	Google RFID .	
2	5710379250208908569	20170731	Google Women's Lightweight Mi...	Google Snapl	
3	5710379250208908569	20170801	Android Hard Cover Journal	Office   Googl	

Results per page: 50 1 – 50 of 111

2. Click **Run** and view the results.

3. **Copy and paste** the below query to explore this public dataset:

Untitled query Run Save Download Share

```

1 SELECT
2   fullVisitorId,
3   date,
4   ARRAY_AGG(v2ProductName) AS products_viewed,
5   ARRAY_AGG(pageTitle) AS pages_viewed
6 FROM `data-to-insights.ecommerce.all_sessions`
7 WHERE visitId = 1501570398
8 GROUP BY fullVisitorId, date
9 ORDER BY date

```

Query completed

Query results Save results Open in

Job information	Results	Chart	JSON	Execution details	Execution graph
Row	fullVisitorId	date	products_viewed	pages_viewed	
1	5710379250208908569	20170731	Google Snapback Hat Black	Google RFID .	
			Google Women's Lightweight Mi...	Google Snapl	
2	5710379250208908569	20170801	Android Hard Cover Journal	Office   Googl	

Results per page: 50 1 – 2 of 2

4. Click **Run** and view the results

5. Next, use the `ARRAY_LENGTH()` function to count the number of pages and products that were viewed:



Query completed

Query results

Job information Results Chart JSON Execution details Execution graph

Row	fullVisitorId	date	products_viewed	num_products_viewed
1	5710379250208908569	20170731	Google Snapback Hat Black	
			Google Women's Lightweight Mi...	
2	5710379250208908569	20170801	Android Hard Cover Journal	

Results per page: 50 1 - 2 of 2

6. Next, deduplicate the pages and products so you can see how many unique products were viewed by adding DISTINCT to ARRAY\_AGG():

Query completed

Query results

Job information Results Chart JSON Execution details Execution graph

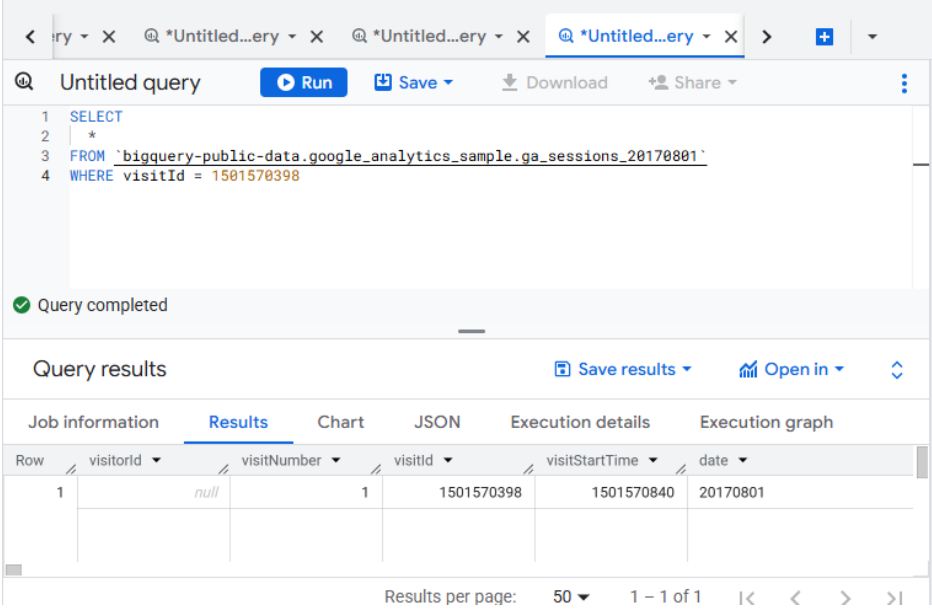
Row	fullVisitorId	date	products_viewed	distinct_products_viewed
1	5710379250208908569	20170731	Google Women's Lightweight Mi...	
			Google Snapback Hat Black	
2	5710379250208908569	20170801	Android Hard Cover Journal	

Results per page: 50 1 - 2 of 2

#### Task 4. Query tables containing arrays

The BigQuery Public Dataset for Google Analytics `bigquery-public-data.google_analytics_sample` has many more fields and rows than our course dataset `data-to-insights.ecommerce.all_sessions`. More importantly, it already stores field values like products, pages, and transactions natively as `ARRAYs`.

1. **Copy and paste** the below query to explore the available data and see if you can find fields with repeated values (arrays):



The screenshot shows the Google Cloud BigQuery console interface. At the top, there's a query editor with the following SQL query:

```
1 SELECT
2 *
3 FROM `bigquery-public-data.google_analytics_sample.ga_sessions_20170801`
4 WHERE visitId = 1501570398
```

Below the query editor, a status bar indicates "Query completed". Underneath, the "Query results" section is displayed, showing a table with the following columns: `Row`, `visitId`, `visitNumber`, `visitId`, `visitStartTime`, and `date`. The table contains one row of data:

Row	visitId	visitNumber	visitId	visitStartTime	date
1	<i>null</i>	1	1501570398	1501570840	20170801

At the bottom of the results section, it shows "Results per page: 50" and "1 - 1 of 1".

2. **Run** the query.
3. Try to query just the visit and page name fields like before:

Untitled query

```
1 SELECT
2   visitId,
3   hits.page.pageTitle
4 FROM `bigquery-public-data.google_analytics_sample.ga_sessions_20170801`
5 WHERE visitId = 1501570398
```

Query completed with errors

Query results

Job information Results Execution details Execution graph

Cannot access field page on a value with type ARRAY<STRUCT<hitNumber INT64, time INT64, hour INT64, ...>> at [3:8]

You will get an error: **Error:**Cannot access field page on a value with type ARRAY<STRUCT<hitNumber INT64, time INT64, hour INT64, ...>> at [3:8]

Untitled query

```
1 SELECT DISTINCT
2   visitId,
3   h.page.pageTitle
4 FROM `bigquery-public-data.google_analytics_sample.ga_sessions_20170801`,
5 UNNEST(hits) AS h
6 WHERE visitId = 1501570398
7 LIMIT 10
```

Query completed

Query results

Job information Results Chart JSON Execution details Execution graph

Row	visitId	pageTitle
1	1501570398	Fun   Accessories   Google Merchandise Store
2	1501570398	Home
3	1501570398	Shop by Brand   Google Merchandise Store
4	1501570398	Office   Google Merchandise Store

We'll cover UNNEST() more in detail later but for now just know that:

- You need to UNNEST() arrays to bring the array elements back into rows
- UNNEST() always follows the table name in your FROM clause (think of it conceptually like a pre-joined table)

## Task 5. Introduction to STRUCTs

You may have wondered why the field alias `hit.page.pageTitle` looks like three fields in one separated by periods. Just as ARRAY values give you the flexibility to *go deep* into the granularity of your fields, another data type allows you to *go wide* in your schema by grouping related fields together. That SQL data type is the **STRUCT** data type.

The easiest way to think about a STRUCT is to consider it conceptually like a separate table that is already pre-joined into your main table.

A STRUCT can have:

- One or many fields in it
- The same or different data types for each field
- It's own alias

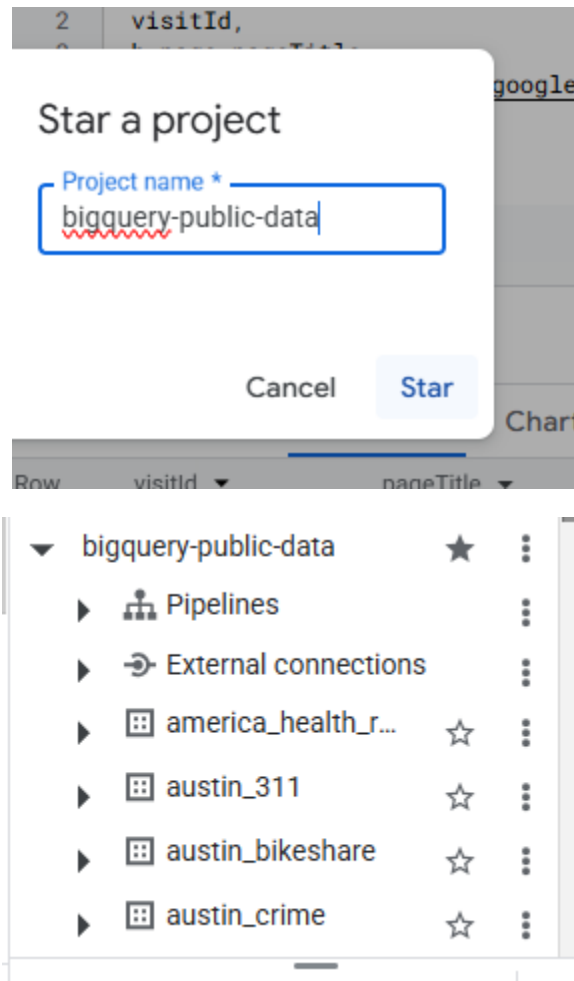
Sounds just like a table right?

Explore a dataset with STRUCTs

1. To open the **bigquery-public-data** dataset, click **+Add Data** and then select **Star a project by name** and enter the name `bigquery-public-data`
2. Click **Star**.

The `bigquery-public-data` project is listed in the Explorer section.

3. Open **bigquery-public-data**.
4. Find and open **google\_analytics\_sample** dataset.
5. Click the **ga\_sessions\_ (366)** table.
6. Start scrolling through the schema and answer the following question by using the find feature of your browser.



The main advantage of having 32 STRUCTs in a single table is it allows you to run queries like this one without having to do any JOINS:

Untitled query

Run Save Download Share

```
1 SELECT
2   visitId,
3   totals.*,
4   device.*
5 FROM `bigquery-public-data.google_analytics_sample.ga_sessions_20170801`
6 WHERE visitId = 1501570398
7 LIMIT 10
```

Query completed

Query results

Save results Open in

Job information Results Chart JSON Execution details Execution graph

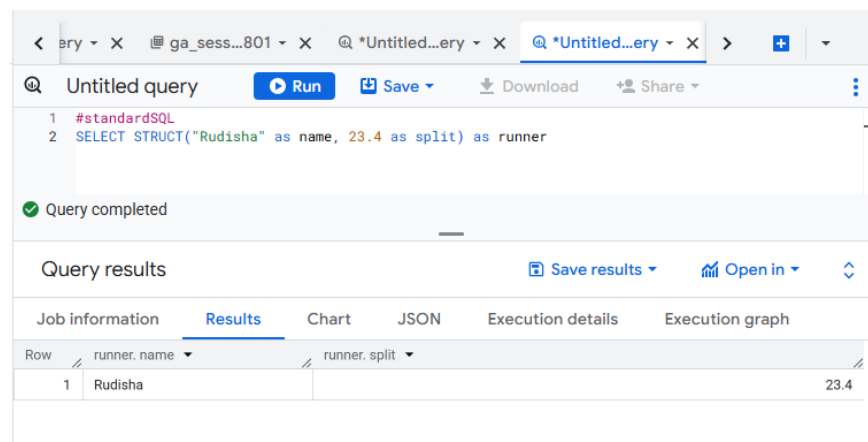
Row	visitId	visits	hits	pageviews	timeOnSite	bounces
1	1501570398	1	11	11	731	

Storing your large reporting tables as STRUCTs (pre-joined "tables") and ARRAYs (deep granularity) allows you to:

- Gain significant performance advantages by avoiding 32 table JOINS
- Get granular data from ARRAYs when you need it but not be punished if you don't (BigQuery stores each column individually on disk)
- Have all the business context in one table as opposed to worrying about JOIN keys and which tables have the data you need

### Task 6. Practice with STRUCTs and arrays

1. With this query, try out the STRUCT syntax and note the different field types within the struct container:



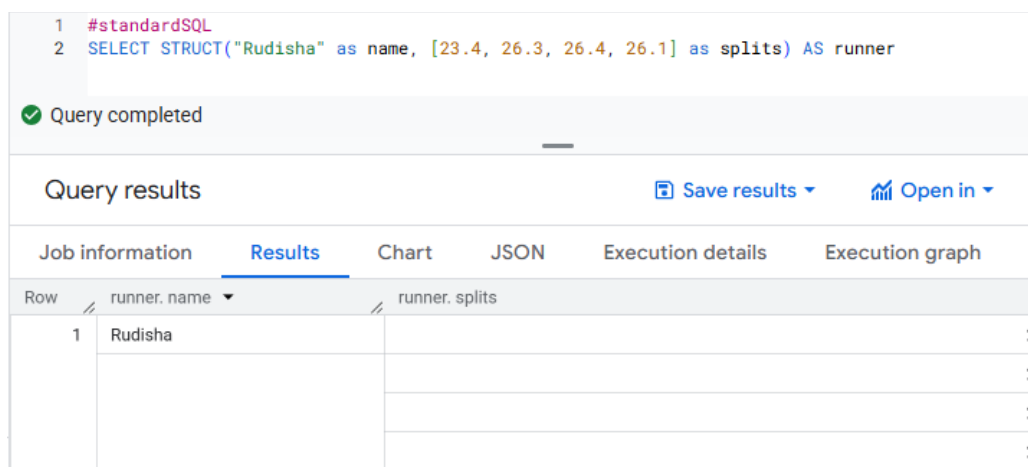
```
1 #standardSQL
2 SELECT STRUCT("Rudisha" as name, 23.4 as split) as runner
```

Query completed

Query results

Row	runner.name	runner.split
1	Rudisha	23.4

2. Run the below query to confirm:



```
1 #standardSQL
2 SELECT STRUCT("Rudisha" as name, [23.4, 26.3, 26.4, 26.1] as splits) AS runner
```

Query completed

Query results

Row	runner.name	runner.splits
1	Rudisha	2
		2
		2
		2

Practice ingesting JSON data

1. Create a new dataset titled racing.

2. Click on racing dataset and click **Create Table**.
3. **Source**: select **Google Cloud Storage** under **Create table from** dropdown.
4. **Select file from Cloud Storage bucket**: cloud-training/data-insights-course/labs/optimizing-for-performance/race\_results.json
5. **File format**: JSONL (Newline delimited JSON)
6. In **Schema**, click on **Edit as text** slider and add the following:

```
[  
  {  
    "name": "race",  
    "type": "STRING",  
    "mode": "NULLABLE"  
  },  
  {  
    "name": "participants",  
    "type": "RECORD",  
    "mode": "REPEATED",  
    "fields": [  
      {  
        "name": "name",  
        "type": "STRING",  
        "mode": "NULLABLE"  
      },  
      {  
        "name": "splits",  
        "type": "FLOAT",  
        "mode": "REPEATED"      }  
    ]  
  }  
]
```

```

    }
  ]
}
]

```

3. Call the new table `race_results`.
4. Click **Create table**.
5. After the load job is successful, preview the schema for the newly created table

The **participants** field is the STRUCT because it is of type RECORD.

The `participants.splits` field is an array of floats inside of the parent `participants` struct. It has a REPEATED Mode which indicates an array. Values of that array are called nested values since they are multiple values inside of a single field.

Create dataset

Project ID \*

qwiklabs-gcp-03-7072cb4c98d9

Change

Dataset ID \*

racing

Letters, numbers, and underscores allowed

Location type ?

Region

Specify a region to colocate your datasets with other Google Cloud services.

Multi-region

Allow BigQuery to select a region within a group to achieve higher quota limits.

Some locations have been restricted due to a policy set by your organization. [Learn more about restricting locations.](#)

Multi-region \*

US (multiple regions in United States)

External Dataset

The selected region supports the following external dataset types: Cloud Spanner

☐ Link to an external dataset ?

Create dataset

Cancel



Create table

×

Source

Create table from

Google Cloud Storage

Select file from GCS bucket or use a URI pattern

cloud-training/data-insights-course/labs/optimizing-for-performance/race\_results.json

Browse

File format

JSONL (Newline delimited JSON)

☐ Source Data Partitioning

Destination

Project

qwiklabs-gcp-03-7072cb4c98d9

Browse

Dataset

racing

Table

Maximum name size is 1,024 UTF-8 bytes. Unicode letters, marks, numbers, connectors, dashes, and spaces are allowed.

Table type

Create table

×

☐ New schema
 

☒ Edit as text

Press Alt+F1 for Accessibility Options.

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

```

{
  "name": "race",
  "type": "STRING",
  "mode": "NULLABLE"
},
{
  "name": "participants",
  "type": "RECORD",
  "mode": "REPEATED",
  "fields": [
    {
      "name": "name",
      "type": "STRING",
      "mode": "NULLABLE"
    },
    {
      "name": "splits",
      "type": "FLOAT",
      "mode": "REPEATED"
    }
  ]
}

```

Create table

Cancel

Explorer

+ Add data

⌵

Search BigQuery resources

?

Show starred only

fruit\_store

fruit\_details

racing

race\_results

Show more

Show more

bigquery-public-data

Repository

Preview

Select a repository and a

race\_results

Query

Open in

Share

Table Explorer

Preview

Insights

Lineage

Data Pri

Filter

Enter property name or value

?

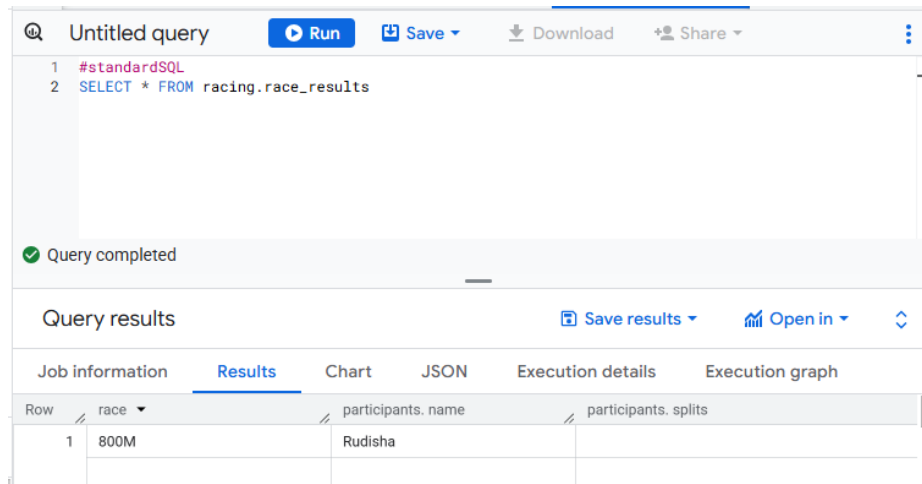
<input type="checkbox"/>	Field name	Type	Mode	Key	Collation	Default Value	Policy Tags
<input type="checkbox"/>	race	STRING	NULLABLE	-	-	-	-
<input type="checkbox"/>	participants	RECORD	REPEATED	-	-	-	-

Edit schema

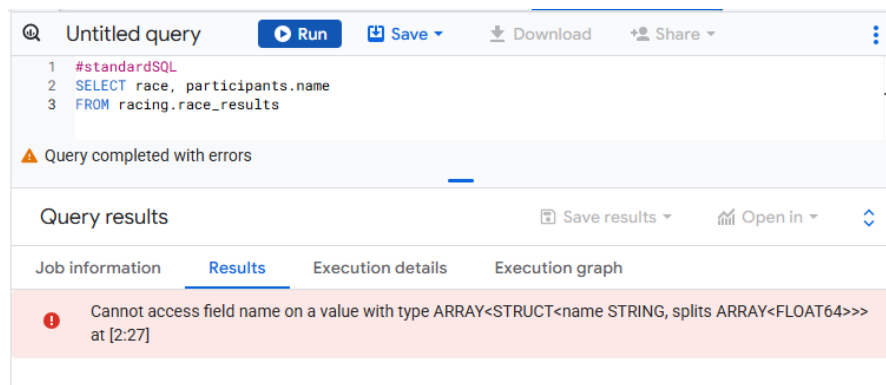
View row access policies

Practice querying nested and repeated fields

- Let's see all of our racers for the 800 Meter race:



2. Run the below schema and see what happens:



**Error:** Cannot access field name on a value with type ARRAY<STRUCT<name STRING, splits ARRAY<FLOAT64>>>> at [2:27]

Much like forgetting to GROUP BY when you use aggregation functions, here there are two different levels of granularity. One row for the race and three rows for the participants names. So how do you change this...

Row	race	participants.name
1	800M	Rudisha
2	???	Makhloufi
3	???	Murphy

...to this:

Row	race	participants.name
1	800M	Rudisha
2	800M	Makhloufi
3	800M	Murphy

In traditional relational SQL, if you had a races table and a participants table what would you do to get information from both tables? You would JOIN them together. Here the participant STRUCT (which is conceptually very similar to a table) is already part of your races table but is not yet correlated correctly with your non-STRUCT field "race".

Can you think of what two word SQL command you would use to correlate the 800M race with each of the racers in the first table?

**Answer:** CROSS JOIN

3. Now try running this:

The screenshot shows a SQL query editor with the following query:

```

1 #standardSQL
2 SELECT race, participants.name
3 FROM racing.race_results
4 CROSS JOIN
5 race_results.participants # full STRUCT name

```

Below the query, a status bar indicates "Query completed".

The "Query results" section shows a table with the following data:

Row	race	name
1	800M	Rudisha
2	800M	Makhloufi
3	800M	Murphy
4	800M	Bosse
5	800M	Rotich

Table name "participants" missing dataset while no default dataset is set in the request.

Even though the participants STRUCT is like a table, it is still technically a field in the racing.race\_results table.

4. Add the dataset name to the query:

Untitled query [Run] [Save] [Download] [Share]

```

1 #standardSQL
2 SELECT race, participants.name
3 FROM racing.race_results AS r, r.participants

```

Query completed

Query results [Save results] [Open in]

Row	race	name
1	800M	Rudisha
2	800M	Makhloufi
3	800M	Murphy
4	800M	Bosse
5	800M	Rotich
6	800M	Lewandowski

5. And click **Run**.

6. You can simplify the last query by:

- Adding an alias for the original table
- Replacing the words "CROSS JOIN" with a comma (a comma implicitly cross joins)

This will give you the same query result

Untitled query [Run] [Save] [Download] [Share]

```

1 #standardSQL
2 SELECT race, participants.name
3 FROM racing.race_results AS r, r.participants

```

Query completed

Query results [Save results] [Open in]

Row	race	name
1	800M	Rudisha
2	800M	Makhloufi
3	800M	Murphy
4	800M	Bosse
5	800M	Rotich
6	800M	Lewandowski

### Task 7. Lab question: STRUCT()

Answer the below questions using the racing.race\_results table you created previously.

**Task:** Write a query to COUNT how many racers were there in total.

- To start, use the below partially written query:

The screenshot shows a query editor with the following SQL query:

```

1 #standardSQL
2 SELECT COUNT(participants.name) AS racer_count
3 FROM racing.race_results

```

Below the query, a message states: "Query completed with errors".

The "Query results" section is visible, showing tabs for "Job information", "Results", "Execution details", and "Execution graph". The "Results" tab is selected.

An error message is displayed in a red box:

```

! Cannot access field name on a value with type ARRAY<STRUCT<name STRING, splits ARRAY<FLOAT64>>>
  at [2:27]

```

### Task 8. Lab question: Unpacking arrays with UNNEST()

Write a query that will list the total race time for racers whose names begin with R. Order the results with the fastest total time first. Use the UNNEST() operator and start with the partially written query below.

- Complete the query:

The screenshot shows a query editor with the following SQL query:

```

1 #standardSQL
2 SELECT
3   p.name,
4   SUM(split_times) as total_race_time
5 FROM racing.race_results AS r
6 , r.participants AS p
7 , p.splits AS split_times
8 WHERE
9 GROUP BY

```

Below the query, a message states: "Query completed with errors".

The "Query results" section is visible, showing tabs for "Job information", "Results", "Execution details", and "Execution graph". The "Results" tab is selected.

An error message is displayed in a red box:

```

! Syntax error: Unexpected keyword GROUP at [9:1]

```

Untitled query	Run	Save	Download	Share
<pre> 2 SELECT 3   p.name, 4   SUM(split_times) as total_race_time 5 FROM racing.race_results AS r 6   , UNNEST(r.participants) AS p 7   , UNNEST(p.splits) AS split_times 8 WHERE p.name LIKE 'R%' 9 GROUP BY p.name 10 ORDER BY total_race_time ASC; </pre>				
Query completed				
Query results				
<div> Job information Results Chart JSON Execution details Execution graph </div>				
Row	name	total_race_time		
1	Rudisha	102.19999999999999		
2	Rotich	103.6		

## Task 9. Filter within array values

You happened to see that the fastest lap time recorded for the 800 M race was 23.2 seconds, but you did not see which runner ran that particular lap. Create a query that returns that result.

- Complete the partially written query:

Untitled query	Run	Save	Download	Share
<pre> 1 #standardSQL 2 SELECT 3   p.name, 4   split_time 5 FROM racing.race_results AS r 6   , UNNEST(r.participants) AS p 7   , UNNEST(p.splits) AS split_time 8 WHERE split_time = 23.2; </pre>				
Query completed				
Query results				
<div> Job information Results Chart JSON Execution details Execution graph </div>				
Row	name	split_time		
1	Kipketer	23.2		

