Loading Data into BigQuery

Setting(create dataset,upload data from csv file)

- Create the *Dataset ID* to **nyctaxi**. Leave the other fields at their default values.

you will load a local CSV into a BigQuery table.
1.Download a subset of the NYC taxi 2018 trips data locally onto your computer from here :
2.In the BigQuery Console, Select the **nyctaxi** dataset then click **Create Table**
Specify the below table options:
Source:
•Create table from: **Upload**
•Choose File: **select the file you downloaded locally earlier**
•File format: **CSV**
Destination:
•Table name: **2018trips** Leave all other setting at default.
Schema:
•Check **Auto Detect** (**tip**: Not seeing the checkbox? Ensure the file format is CSV and not Avro)
Advanced Options
•Leave at default values

Click **Create Table**.
3.You should now see the **2018trips** table below the nyctaxi dataset.
Select the 2018trips table and view **details**:
4.Select **Preview** and confirm all columns have been loaded (sampled below):
You have successfully loaded in a CSV file into a new BigQuery table.

Setting(upload from Google Cloud Storage)

Now, lets try load another subset of the same 2018 trip data that is available on Cloud Storage.
And this time, let's use the CLI tool to do it.
1.In your **Cloud Shell**, run the following command :

```
bq load \
--source_format=CSV \
--autodetect \
--noreplace  \
nyctaxi.2018trips \
gs://cloud-training/OCBL013/nyc_tlc_yellow_trips_2018_subset_2.csv
```

Creating table using DDL

```
#standardSQL
CREATE TABLE
  nyctaxi.january_trips AS
SELECT
  *
FROM
  nyctaxi.2018trips
WHERE
  EXTRACT(Month
  FROM
    pickup_datetime)=1;
```

Lab :
Working with JSON
and
Array data in BigQuery

Intro to ARRAY in Bigquery

Create dataset fruit_store.
In traditional **relational database SQL**, we use normalization (going from one table to many).
For **data warehousing**, data analysts often go the reverse direction (denormalization) and bring many separate tables into one large reporting table.

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

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

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

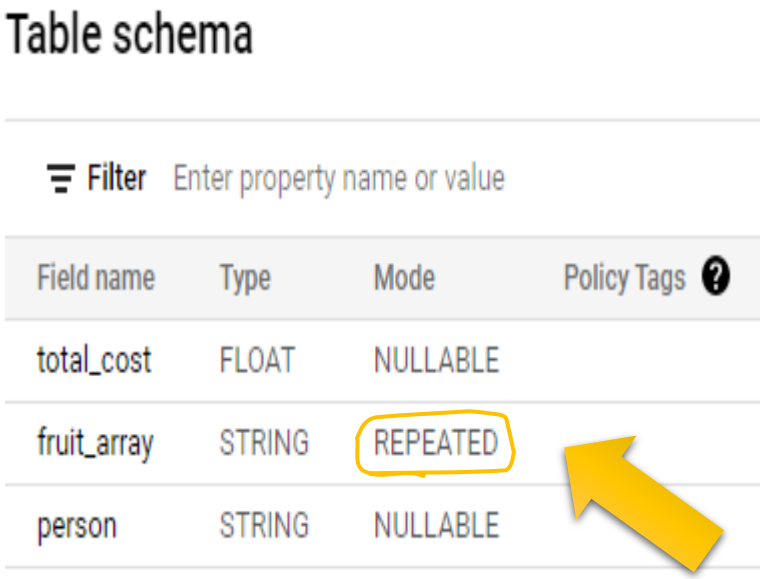
Explore ARRAY in Bigquery,showing result as json

- SELECT ['raspberry', 'blackberry', 'strawberry', 'cherry'] AS fruit_array
- SELECT ['raspberry', 'blackberry', 'strawberry', 'cherry', 1234567] AS fruit_array ✘ (same type)
- **SELECT person, fruit_array, total_cost FROM `data-to-insights.advanced.fruit_store`;**

Loading semi-structured JSON into BigQuery

- Create a new table in the **fruit_store** dataset twith following details :
  - **Source**: Choose **Google Cloud Storage** in the **Create table from** dropdown.
  - **Select file from GCS bucket** (type or paste the following): data-insights-course/labs/optimizing-for-performance/shopping_cart.json
  - **File format**: JSONL (Newline delimited JSON) {This will be auto-populated}
  - **Schema**: Check **Auto detect** (Schema and input parameters).
- Call the new table "fruit_details".

.



In the schema, note that fruit_array is marked as **REPEATED** which means it's an array.

Recap

- BigQuery natively supports arrays
- Array values must share a data type
- Arrays are called REPEATED fields in BigQuery

Creating your own arrays with ARRAY_AGG(), ARRAY_LENGTH()

```
SELECT
  fullVisitorId,
  date,
  v2ProductName,
  pageTitle
  FROM `data-to-insights.ecommerce.all_sessions`
WHERE visitId = 1501570398
ORDER BY date
```

111 rows

```
SELECT
  fullVisitorId,
  date,
  ARRAY_AGG(v2ProductName) AS products_viewed,
  ARRAY_AGG(pageTitle) AS pages_viewed
  FROM `data-to-insights.ecommerce.all_sessions`
WHERE visitId = 1501570398
GROUP BY fullVisitorId, date
ORDER BY date
```

2 - one for each day

```
SELECT
  fullVisitorId,
  date,
  ARRAY_AGG(v2ProductName) AS products_viewed,
  ARRAY_LENGTH(ARRAY_AGG(v2ProductName)) AS num_products_viewed,
  ARRAY_AGG(pageTitle) AS pages_viewed,
  ARRAY_LENGTH(ARRAY_AGG(pageTitle)) AS num_pages_viewed
  FROM `data-to-insights.ecommerce.all_sessions`
WHERE visitId = 1501570398
GROUP BY fullVisitorId, date
ORDER BY date
```

Row	fullVisitorId	date	products_viewed	num_products_viewed	pages_viewed	num_pages_viewed
1	5710379250208908569	20170731	(2 rows)	2	(2 rows)	2
2	5710379250208908569	20170801	(10+ rows)	109	(10+ rows)	109

Querying datasets that already have ARRAYs , query on repeated field

```
SELECT
  *
FROM `bigquery-public-
data.google_analytics_sample.ga_sessions_20170801`
WHERE visitId = 1501570398
```

Scroll right in the results until you see the hits.product.v2ProductName field (we will discuss the multiple field aliases shortly).

SELECT **visitId,** **hits.page.pageTitle** **FROM `bigquery-public-data.google_analytics_sample.ga_sessions_20170801`** **WHERE visitId = 1501570398**	**SELECT DISTINCT** **visitId,** **h.page.pageTitle** **FROM `bigquery-public-data.google_analytics_sample.ga_sessions_20170801`,** **UNNEST(hits) AS h** **WHERE visitId = 1501570398** **LIMIT 10**
You will get an error: Cannot access field product on a value with type ARRAY> at [5:8]	How do we do that with SQL? Answer: Use the UNNEST() function on your array field:
Before we can query REPEATED fields (arrays) normally, you must first break the arrays back into rows. For example, the array for hits.page.pageTitle is stored currently as a single row like: **['homepage','product page','checkout']** and we need it to be **['homepage',** **'product page',** **'checkout']**	You need to UNNEST() arrays to bring the array elements back into rows

STRUCTs

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

Add public dataset **ga_sessions , you find struct fields e.g.**
Totals , TrafficSource , device , trafficSource.adwordsClickInfo

```
SELECT
 visitId,
 totals.*,
 device.*
FROM `bigquery-public-data.google_analytics_sample.ga_sessions_20170801`
WHERE visitId = 1501570398
LIMIT 10
```

•get granular data from ARRAYs when you need it but not be punished if you don't (BigQuery stores each column individually on disk)
•NO joins ➔ high performance
•have all the business context in one table as opposed to worrying about JOIN keys and which tables have the data you need

Practice with STRUCTs and ARRAYs

```
SELECT STRUCT("Rudisha" as name, 23.4 as split) as runner
```

Row	runner.name	runner.split
1	Rudisha	23.4

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

Row	runner.name	runner.splits
1	Rudisha	23.4
		26.3
		26.4
		26.1

- Structs are containers that can have multiple field names and data types nested inside.
- An arrays can be one of the field types inside of a Struct (as shown above with the splits field).

Practice ingesting JSON data

1.Create a new dataset titled **racing**.
2.Create a new table titled **race_results**.
3.Ingest this Google Cloud Storage JSON file: **data-insights-course/labs/optimizing-for-performance/race_results.json**
•**Source**: Google Cloud Storage under **Create table from** dropdown.
•**Select file from GCS bucket**: data-insights-course/labs/optimizing-for-performance/race_results.json
•**File format**: JSONL (Newline delimited JSON)
•In **Schema**, move the **Edit as text** slider and add the following
•Preview table and discover structs and arrays

```
[
  {
    "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"
      }
    ]
  }
]
```

Queries

1- Let's see all of our racers for the 800 Meter race
#standardSQL
SELECT * FROM racing.race_results

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
			26.0
			27.0
			26.0
		Bosse	23.6

2.What if you wanted to list the name of each runner and the type of race?
Run the below schema and see what happens:

```
#standardSQL
SELECT race, participants.name
FROM racing.race_results
```

Error: Cannot access field name on a value with type ARRAY\<STRUCT\<name STRING, splits ARRAY\<FLOAT64\>>>> at [1:21]

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

Answer: CROSS JOIN

...to this:

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

```
#standardSQL
SELECT race, participants.name
FROM racing.race_results
CROSS JOIN
participants  # this is the STRUCT (it's like a table within a table)
```

Error: Table name "participants" cannot be resolved: dataset name is missing.

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

```
#standardSQL
SELECT race, participants.name
FROM racing.race_results
CROSS JOIN
race_results.participants # full STRUCT name
```

```
# another solution
#standardSQL
SELECT race, participants.name
FROM racing.race_results AS r,
r.participants
```

Row	race	name
1	800M	Rudisha
2	800M	Makhloufi
3	800M	Murphy
4	800M	Bosse
5	800M	Rotich
6	800M	Lewandowski
7	800M	Kipketer
8	800M	Berian

Lab Question: STRUCT()

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

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

Row	racer_count
1	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

```
#standardSQL
SELECT
  p.name,
  SUM(split_times) as total_race_time
FROM racing.race_results AS r
, UNNEST(r.participants) AS p
, UNNEST(p.splits) AS split_times
WHERE p.name LIKE 'R%'
GROUP BY p.name
ORDER BY total_race_time ASC;
```

Row	name	total_race_time
1	Rudisha	102.199999999999
2	Rotich	103.6

Lab Question: Filtering 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

```
#standardSQL
SELECT
  p.name,
  split_time
FROM racing.race_results AS r
, UNNEST(r.participants) AS p
, UNNEST(p.splits) AS split_time
WHERE split_time = 23.2;
```

Row	name	split_time
1	Kipketer	23.2