Modeling BigQuery Nested Structure in Looker

Lan Tran (lantrann@google.com)

November 2020

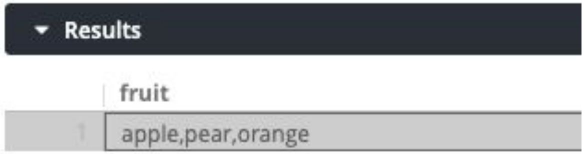
Section 1

- Datatype: ARRAY[], STRUCT()
- Function: FLATTEN(), and UNNEST()

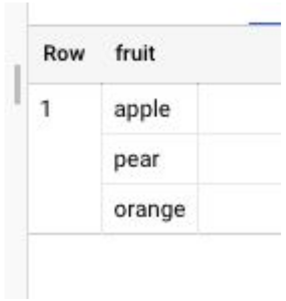
ARRAY: An object contains multiple elements of the same datatype in BQ

BigQuery requires that an array must contain elements that have the same data type BQ Doc

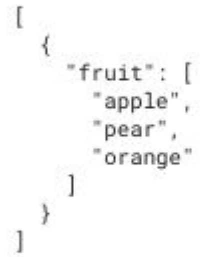
SELECT ["apple", "pear", "orange"] as fruit
# `fruit` is a string array



	fruit
1	apple,pear,orange



Row	fruit
1	apple
	pear
	orange



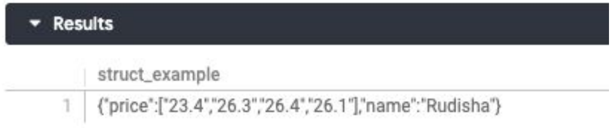
```
[
  {
    "fruit": [
      "apple",
      "pear",
      "orange"
    ]
  }
]
```

Using StandardSQL in BigQuery will automatically "FLATTEN" the value (more details below). The JSON result keeps the string value inside an array.

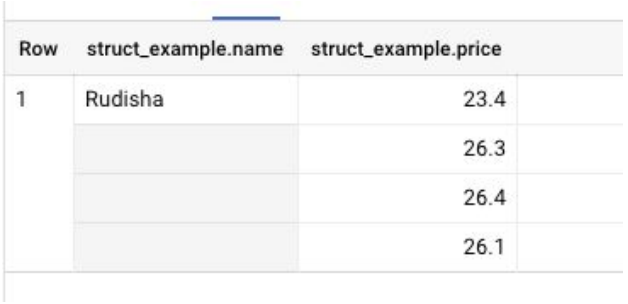
STRUCT: A flexible container of multiple elements

A struct is a flexible container of multiple elements, and these elements do not need to be the same datatype. For example, a struct can contain a string, an integer array, and a string array.

```
SELECT STRUCT("Rudisha" as name, [23.4, 26.3, 26.4, 26.1] as price) as struct_example
# `struct_example` is a struct containing a string `struct_example.name`, and a float array, `struct_example.splits`
```



	struct_example
1	{"price":["23.4","26.3","26.4","26.1"],"name":"Rudisha"}



Row	struct_example.name	struct_example.price
1	Rudisha	23.4
		26.3
		26.4
		26.1

NOTE: the element inside a struct needs to be fully scoped to be accessed with Standard SQL

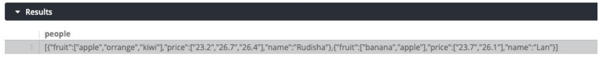
```
SELECT struct_example.name
FROM table
```

REPEATED FIELDS: Arrays of structs

A repeated field is an array of structs

```
SELECT
   [STRUCT("Rudisha" as name, ["apple", "orrange", "kiwi"] as fruit, [23.2, 26.7, 26.4] as price),
    STRUCT("Lan" as name, ["banana", "apple"] as fruit, [23.7, 26.1] as price)]
   AS people
```

\`people\` is an array that contains two structs. Each struct contains a string (\`people.name\`), a string array (\`people.fruit\`), and an float array (\`people.price\`). \`people\` is also considered a **repeated field** as we see \`people\` has two \`records\` (one for "Rudisha" and one for "Lan").



```
   people
1  [{"fruit":["apple","orrange","kiwi"],"price":["23.2","26.7","26.4"],"name":"Rudisha"},{"fruit":["banana","apple"],"price":["23.7","26.1"],"name":"Lan"}]
```

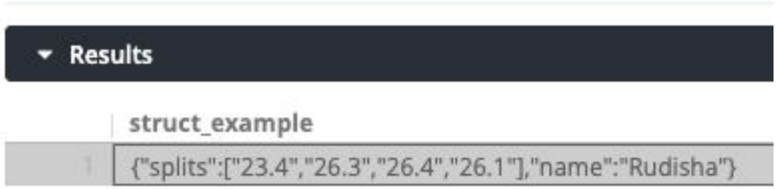
people	RECORD	REPEATED
people.**name**	STRING	NULLABLE
people.**fruit**	STRING	NULLABLE
people.**price**	INTEGER	NULLABLE

Row	people.name	people.fruit	people.price
1	Rudisha	apple	23.2
		orrange	26.7
		kiwi	26.4
	Lan	banana	23.7
		apple	26.1

FLATTEN: Natively-supported tables in BQ

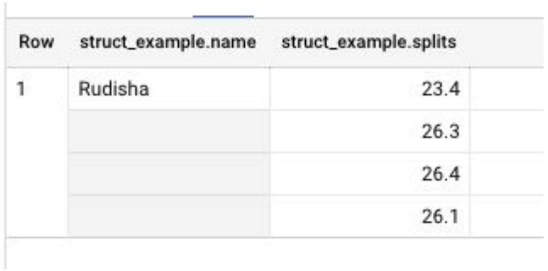
StandardSQL in BigQuery will implicitly FLATTEN an array without the need to include the function FLATTEN() in SQL syntax. There is no FLATTEN() function in BigQuery.

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



	struct_example
1	{"splits":["23.4","26.3","26.4","26.1"],"name":"Rudisha"}

In Looker: 1 row with value stored inside JSON



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

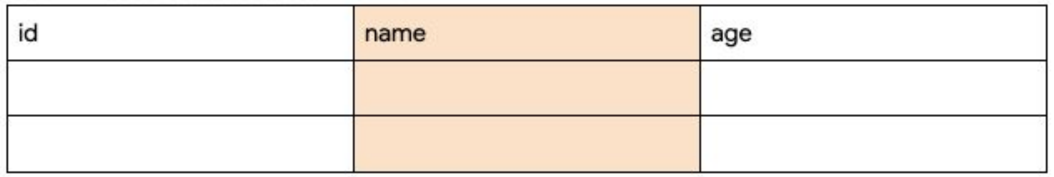
In BigQuery: The values inside `struct_example.split` are flattened

UNNEST (1): Row-based vs. Column-based databases

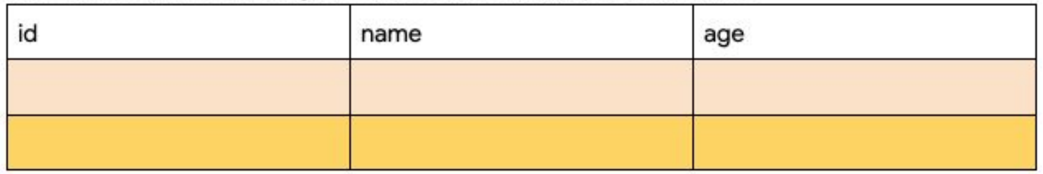
To understand UNNEST() in BigQuery, we need to understand the difference between row-based databases (MySQL, Postgres, etc.) and column-based databases (BigQuery, Redshift, Snowflake, etc.).

Considering this query: `SELECT name FROM table WHERE name = ‘John’`

A column-based database (such as BigQuery) only needs to bring the \`name\` column into memory and scans that column to find the value “John”. A row-oriented database will bring the entire row into memory in order to scan the \`name\` column.


Column-based database scan \`name\`

id	name	age


Row-based databases bring the entire row to memory to scan \`name\`

id	name	age

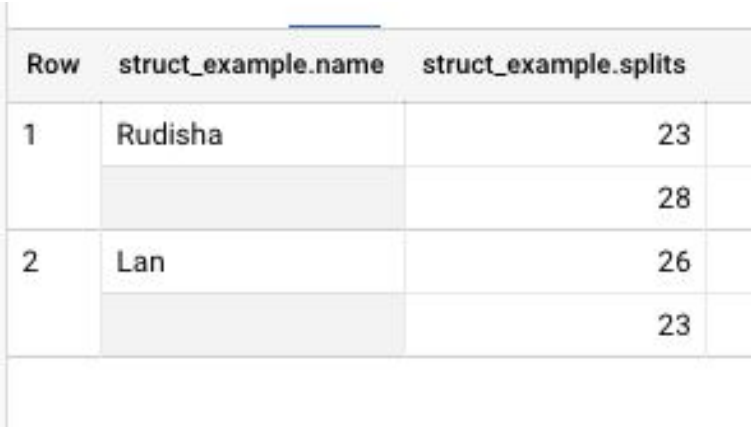
UNNEST (2): Understand a simple table with nested data

Proprietary

Let's look at a simple example with UNNEST in BigQuery.

```
WITH table as
(SELECT STRUCT("Rudisha" as name, [23, 28] as splits) as struct_example
UNION ALL
SELECT STRUCT("Lan" as name, [26, 23] as splits) as struct_example)
SELECT * FROM table
```

In `table`, we have two rows, each is a struct, `struct_example`, containing a string (`struct_example.name`) and an integer array (`struct_example.splits`)



Row	struct_example.name	struct_example.splits
1	Rudisha	23
		28
2	Lan	26
		23

Google

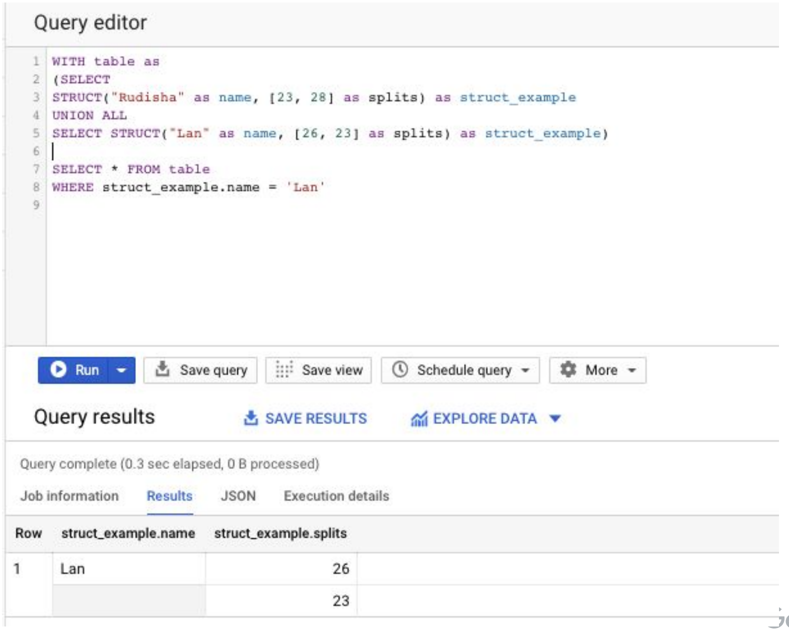
UNNEST (3): A struct does not need to be unnested

Case 1: I want to find all of the records `WHERE struct_example.name = "Lan"`.

Since "Lan" as a value is already stored in the column `struct_example.name`, BigQuery only needs to scan that column to find "Lan"

```
WITH table as
(SELECT
STRUCT("Rudisha" as name, [23, 28] as splits) as struct_example
UNION ALL
SELECT STRUCT("Lan" as name, [26, 23] as splits) as
struct_example)

SELECT * FROM table
WHERE struct_example.name = 'Lan'
```



```
WITH table as
(SELECT
STRUCT("Rudisha" as name, [23, 28] as splits) as struct_example
UNION ALL
SELECT STRUCT("Lan" as name, [26, 23] as splits) as struct_example)

SELECT * FROM table
WHERE struct_example.name = 'Lan'
```

Query complete (0.3 sec elapsed, 0 B processed)

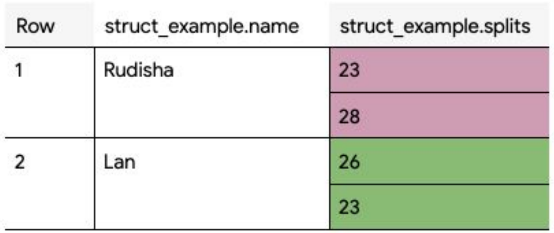
Row	struct_example.name	struct_example.splits
1	Lan	26
		23

UNNEST (4): Values inside an array of a repeated record needs to be unnested

Case 2: I want to find all records `WHERE struct_example.splits = 23`.

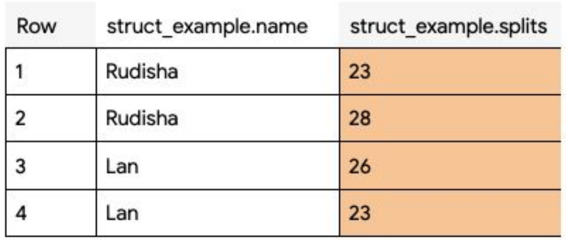
If we write a syntax using the same logic above, SELECT * FROM table WHERE struct_example.splits = 23, we immediately get the error `No matching signature for operator = for argument types: ARRAY<INT64>, STRING. Supported signature: ANY = ANY at [5:7]`

`23` is not available for BigQuery to scan because it is still inside an array (`struct_example.splits`). When BigQuery scans the column `struct_example.splits`, all it could see are the two containers “pink” and “green” - there is no `23` to see (because `23` is still inside the container “pink”)



Row	struct_example.name	struct_example.splits
1	Rudisha	23
		28
2	Lan	26
		23

What we need to do is to unnest the array so the value `23` is available to be scanned.

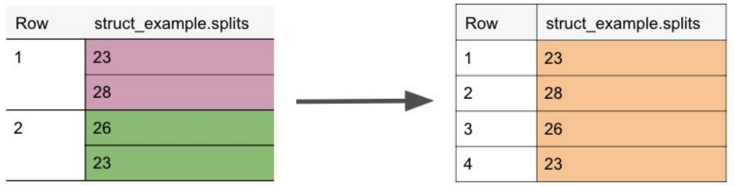


Row	struct_example.name	struct_example.splits
1	Rudisha	23
2	Rudisha	28
3	Lan	26
4	Lan	23

UNNEST (5): Breaking down the syntax CROSS JOIN UNNEST()

```
SELECT struct_example.name, s
FROM table CROSS JOIN UNNEST(struct_example.splits) as s
```

`UNNEST(struct_example.splits) as s` will make a table called `s` with a single row for each element in `struct_example.splits`

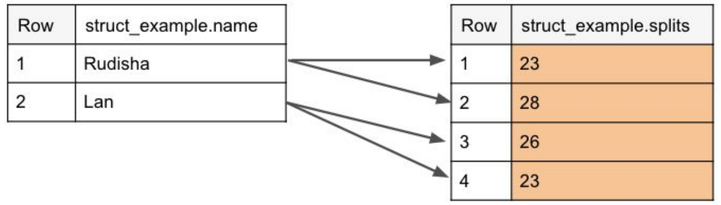


Row	struct_example.splits
1	23
	28
2	26
	23

Row	struct_example.splits
1	23
2	28
3	26
4	23

CROSS JOIN UNNEST(struct_example.splits) as s

CROSS JOIN UNNEST(field) will match an array with the respective value of the array in the same row. In the example above, “Lan” and the array [26,23] are on the same record. Therefore, when we CROSS JOIN UNNEST, “Lan” is cross-joined with [26,23] and not [23,28]



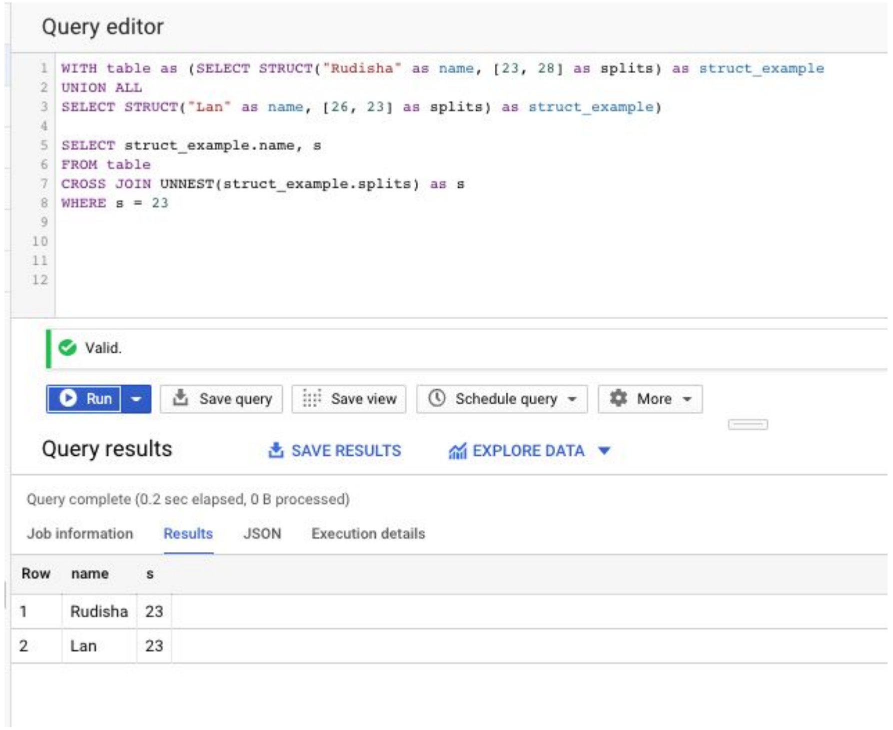
Row	struct_example.name
1	Rudisha
2	Lan

Row	struct_example.splits
1	23
2	28
3	26
4	23

UNNEST (6): Putting things together

Now that everything is flattened, all values are available to be scanned

Row	struct_example.name	struct_example.splits
1	Rudisha	23
2	Rudisha	28
3	Lan	26
4	Lan	23



```sql
WITH table as (SELECT STRUCT("Rudisha" as name, [23, 28] as splits) as struct_example
UNION ALL
SELECT STRUCT("Lan" as name, [26, 23] as splits) as struct_example)

SELECT struct_example.name, s
FROM table
CROSS JOIN UNNEST(struct_example.splits) as s
WHERE s = 23
```

Query complete (0.2 sec elapsed, 0 B processed)

Row	name	s
1	Rudisha	23
2	Lan	23

UNNEST (7): Using ARRAY() to nest back

We can take additional steps to transform the result back to a nested field. Since nested fields are not natively supported by Looker, I'll skip it here. If you want to learn how to transform the results back to an array, look at the two functions in BigQuery.

ARRAY_AGG()

ARRAY_LENGTH()

Section 2

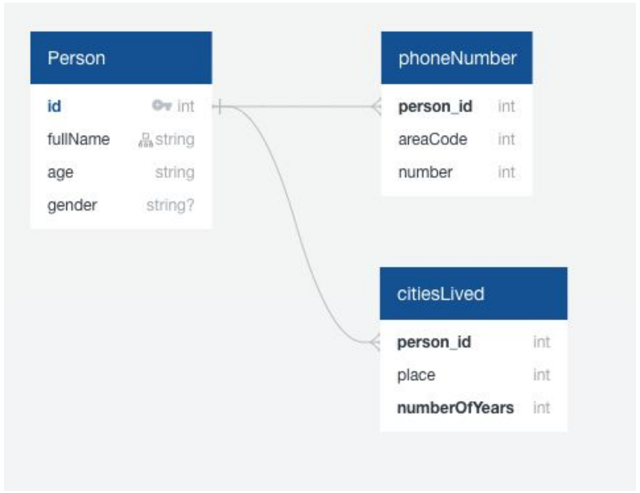
Using LookML on Nested Fields / Repeated Records / Non-repeated Records

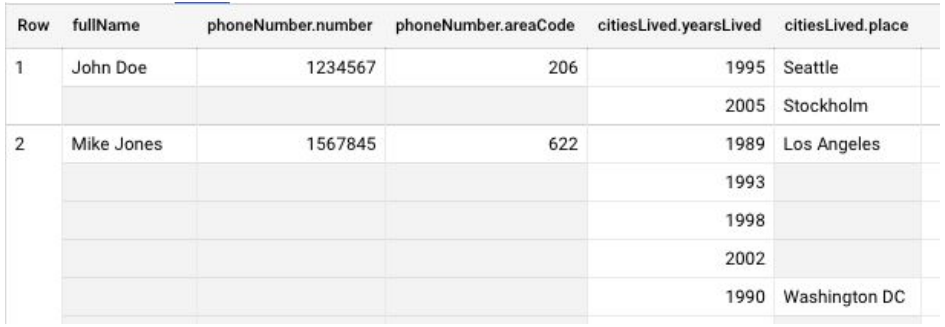
Nested Data: Less joins needed = More performant-query

We have an article written by Lloyd "Nested Data in BigQuery (Repeated Records)". The next following slides will break down the article.

A traditional relational database may lay out the data structure with 3 tables "person", "phoneNumber" and "citiesLived". When we want to extract all details of a person, we join the three tables together. In large databases, joining multiple tables with millions of records can be performant heavy.

BigQuery introduces a neat concept called "Nested Data". Instead of having 3 separate tables: "person", "phoneNumber" and "citiesLived", we now join "phoneNumber" and "citiesLived" into "person"





Row	fullName	phoneNumber.number	phoneNumber.areaCode	citiesLived.yearsLived	citiesLived.place
1	John Doe	1234567	206	1995	Seattle
				2005	Stockholm
2	Mike Jones	1567845	622	1989	Los Angeles
				1993	
				1998	
				2002	
				1990	Washington DC

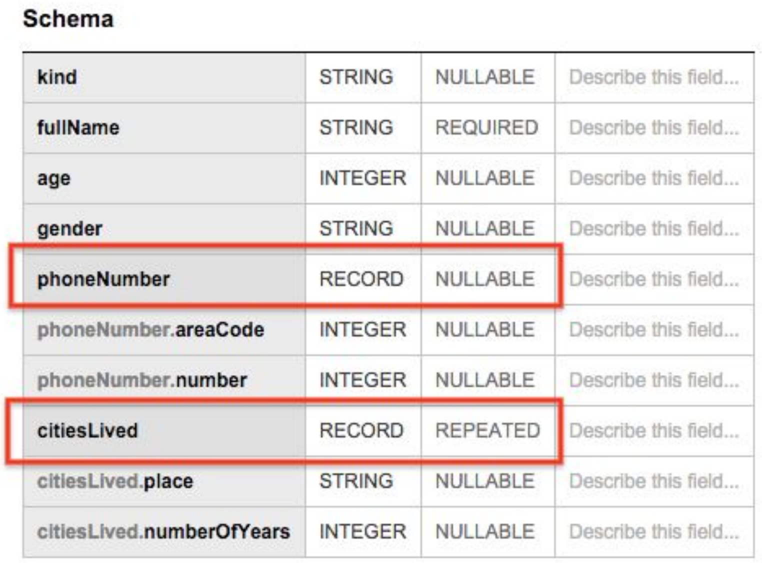
Define LookML views: Primary table & Records

Looking at the schema, we recognize there are three tables: “person” (the primary), “phoneNumber”, and “citiesLived”. Each table should be made into its own view in LookML. We define 3 views in LookML. Inside each view, we define all dimensions. In the primary view, “record” dimensions should be hidden. When we only select the record dimension, it will return data in JSON, and when we select it with another field, we will encounter the error Query execution failed: - Grouping by expressions of type ARRAY is not allowed at [7:10] as Looker automatically GROUP BY all dimensions.

```
view: persons {
  dimension: age {}
  dimension: phoneNumber {hidden: yes}
  dimension: citiesLive {hidden: yes}
 }

view: persons_phone_number {
  dimension: areaCode {}
  dimension: number {}
}

view: persons_cities_lived {
  dimension: place {}
  dimension: numberOfYears {}
}
```



Schema

kind	STRING	NULLABLE	Describe this field...
fullName	STRING	REQUIRED	Describe this field...
age	INTEGER	NULLABLE	Describe this field...
gender	STRING	NULLABLE	Describe this field...
phoneNumber	RECORD	NULLABLE	Describe this field...
phoneNumber.areaCode	INTEGER	NULLABLE	Describe this field...
phoneNumber.number	INTEGER	NULLABLE	Describe this field...
citiesLived	RECORD	REPEATED	Describe this field...
citiesLived.place	STRING	NULLABLE	Describe this field...
citiesLived.numberOfYears	INTEGER	NULLABLE	Describe this field...

Define LookML explore (1): Join the repeated nested object

```
explore: persons {

# Repeated nested object
join: persons_cities_lived {
view_label: "Persons: Cities Lived:"
sql: LEFT JOIN UNNEST(persons.citiesLived) as
persons_cities_lived ;;
relationship: one_to_many
}


}
```

Type of join: CROSS JOIN UNNEST() will remove rows with null rows while LEFT JOIN UNNEST() will maintain the null rows in the table. We define the join using `sql` (Quick refresh: We use sql_on for join but this is a special concept, so we need to use `sql` -- sql for join doc)

Type of relationship: “one-to-many” for repeated records.

Define LookML explore (2): Join the non-repeated nested object

```
explore: persons {
# Non repeated nested object
join: persons_phone_number {
view_label: "Persons: Phone:"
sql: LEFT JOIN UNNEST([${persons.phoneNumber}])
as persons_phone_number ;;
relationship: one_to_one
}
}
```

APPROACH 1 (described in Lloyd's article):

Type of join: Notice that for a non repeated nested object, we are surrounding it with [ ] to turn it into an array. phoneNumber is a nested object but it's not a repeated field so we can't unnest it just yet. *Only arrays can be unnested.* Thus, we can turn that object into an array by wrapping it in brackets, as [ ] is an indication for an array.

Type of relationship: "one-to-one" for non-repeated records.

```
Explore: Persons {}

view: Persons {
  dimension: PhoneNumber {
    sql: ${TABLE}.persons.phoneNumber ;;
  }
}
```

APPROACH 2:

Since Standard SQL in BigQuery assumes explicit "FLATTEN", the value inside a non-repeated record, can be accessed directly in the main view. With that said, we do not need to make a separate view and we can define the non-repeated field directly in the main view.

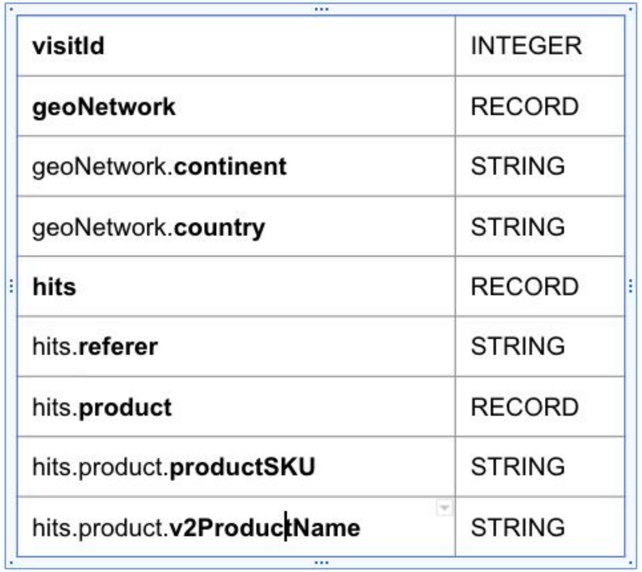
PRACTICE

Dataset: We will use selected columns from a table in the public dataset provided by BigQuery. We can query this database using BQ console or SQL Runner to any BigQuery connection. I'll use the `connection:

`bigquery-public-data.google_analytics_sample.ga_sessions_20170801`

Refer to this data dictionary for the full description

To the right is a preview of the schema (with the majority of fields omitted for simplicity).



visitId	INTEGER
geoNetwork	RECORD
geoNetwork.**continent**	STRING
geoNetwork.**country**	STRING
hits	RECORD
hits.**referer**	STRING
hits.**product**	RECORD
hits.product.**productSKU**	STRING
hits.product.**v2ProductName**	STRING

Build a LookML model with nested data

Build a LookML model to answer the following questions and send the explore link to finish the exercises:

1. Find the top 10 products that are accessed through Google Canada (WHERE referer CONTAINS google.ca)
2. Find the top 10 countries that search for “Home/Bags/” (WHERE V2ProductName = “Home/Bags”)

Hint: The table includes double-nested fields. If we want to access `hits.product.productSKU`, we need to UNNEST two levels, first from MAIN > HITS, then HITS > PRODUCT). Think about how you would write the join in the explore level.

Additional Resources

- The Google Analytics block on Looker's Marketplace is using the same data structure that we use for our practice. You can read their model and view files for further inspiration: Marketplace GA block modeling file
- Easy JSON Path finder: https://jsonpathfinder.com/
- JSON Generator: https://www.json-generator.com/
- What is JSON https://www.w3schools.com/whatis/whatis_json.asp