Writing queries in BigQuery

INTRODUCTION TO BIGQUERY

Matt Forrest
Field CTO

Writing simple queries

A simple query in BigQuery

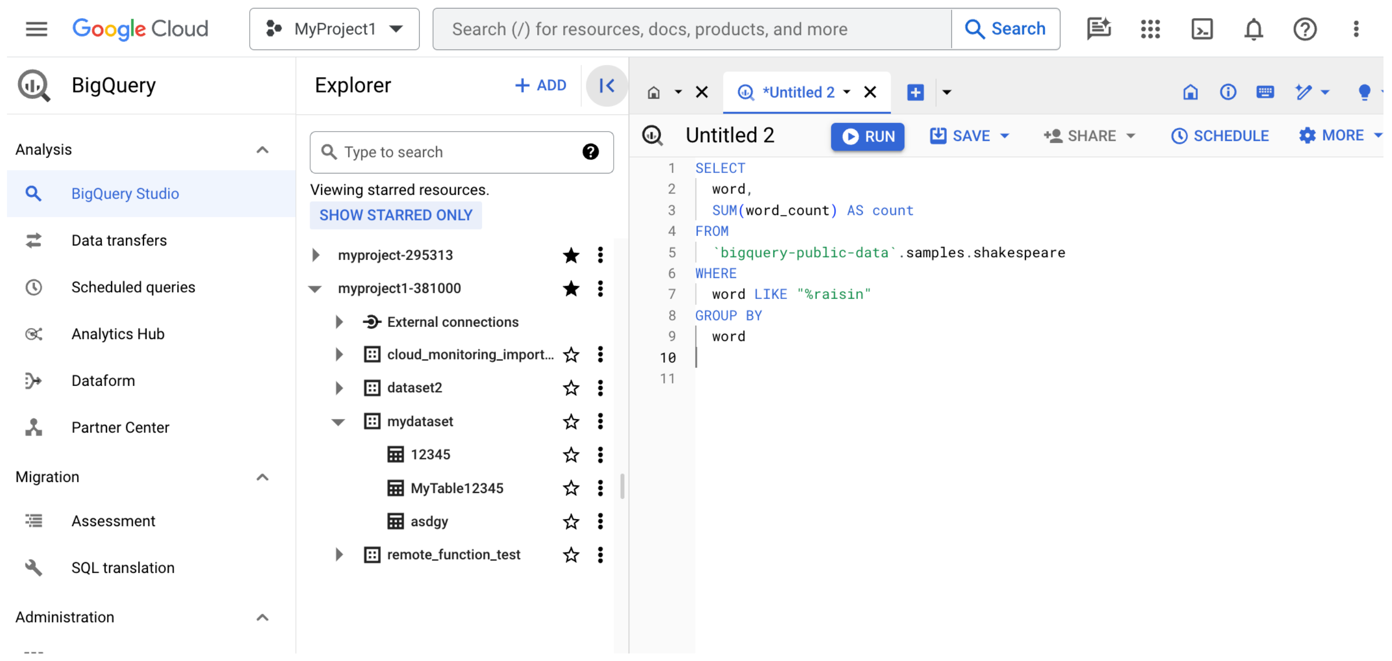
```
-- Note the table name structure

SELECT
    *
FROM
    `project.ecommerce.order_items`
```

Running queries in BigQuery

We can run queries in BigQuery via:

- BigQuery Studio
- Client libraries (e.g., Python)
- Google Cloud command line tool
- Pandas



```sql
SELECT
  word,
  SUM(word_count) AS count
FROM
  `bigquery-public-data`.samples.shakespeare
WHERE
  word LIKE "%raisin"
GROUP BY
  word
```

Using correct table names

```sql
-- Using the full table name structure


SELECT
    *
FROM
    `dataset.ecommerce.order_items`
```

```sql
/* Using the shorthand
table name structure */

SELECT
    *
FROM
    ecommerce.order_items
```

GoogleSQL

Legacy SQL	GoogleSQL	Notes
BOOL	BOOL	
INTEGER	INT64	
FLOAT	FLOAT64	
STRING	STRING	
BYTES	BYTES	
RECORD	STRUCT	

Our datasets: Olist E-Commerce

- **Orders:** Order number and order item information
- **Order details:** Customer id, order and shipping dates
- **Payments:** Payment type, split payments, amounts
- **Products:** Product category, description, dimensions

[1] https://www.kaggle.com/datasets/olistbr/brazilian-ecommerce

Products

- `product_id` - unique product ID
- `product_photos_qty` - number of product photos
- `product_weight_g` - weight of the product
- `product_category_name_english` - product category name

Orders

- `order_id` - Order unique ID
- `order_items` - `STRUCT` containing information about the order items
  - `order_item_id` - Item number in the order
  - `product_id` - Unique product ID
  - `seller_id` - Unique seller ID
  - `price` - Price of the order item

Order details

- `order_id` - unique order ID
- `customer_id` - unique customer ID
- `order_status` - current order status
- `order_purchase_timestamp` - Timestamp when order was purchased
- `order_approved_at` - Timestamp when order was approved
- `order_delivered_carrier_date` - Timestamp when order was accepted by the carrier
- `order_delivered_customer_date` - Timestamp when order delivered
- `order_estimated_delivery_date` - Timestamp of estimated delivery date

Payments

- `order_id` - unique order ID
- `payment_type` - type of payment
- `payment_sequential` - payment number
- `payment_installments` - number of payment installments
- `payment_value` - value of that payment

Review of aggregations and joins

```
-- Count of orders per customers


SELECT
    d.customer_id,
    COUNT(o.order_id)
FROM
    dataset.order_details d
JOIN
    dataset.orders o
USING (order_id)
GROUP BY
    d.customer_id
```

Five key components:

1. Aggregate function
2. The left dataset
3. The right dataset
4. Join condition
5. Grouping condition

Let's practice!

INTRODUCTION TO BIGQUERY

Data ingestion in BigQuery

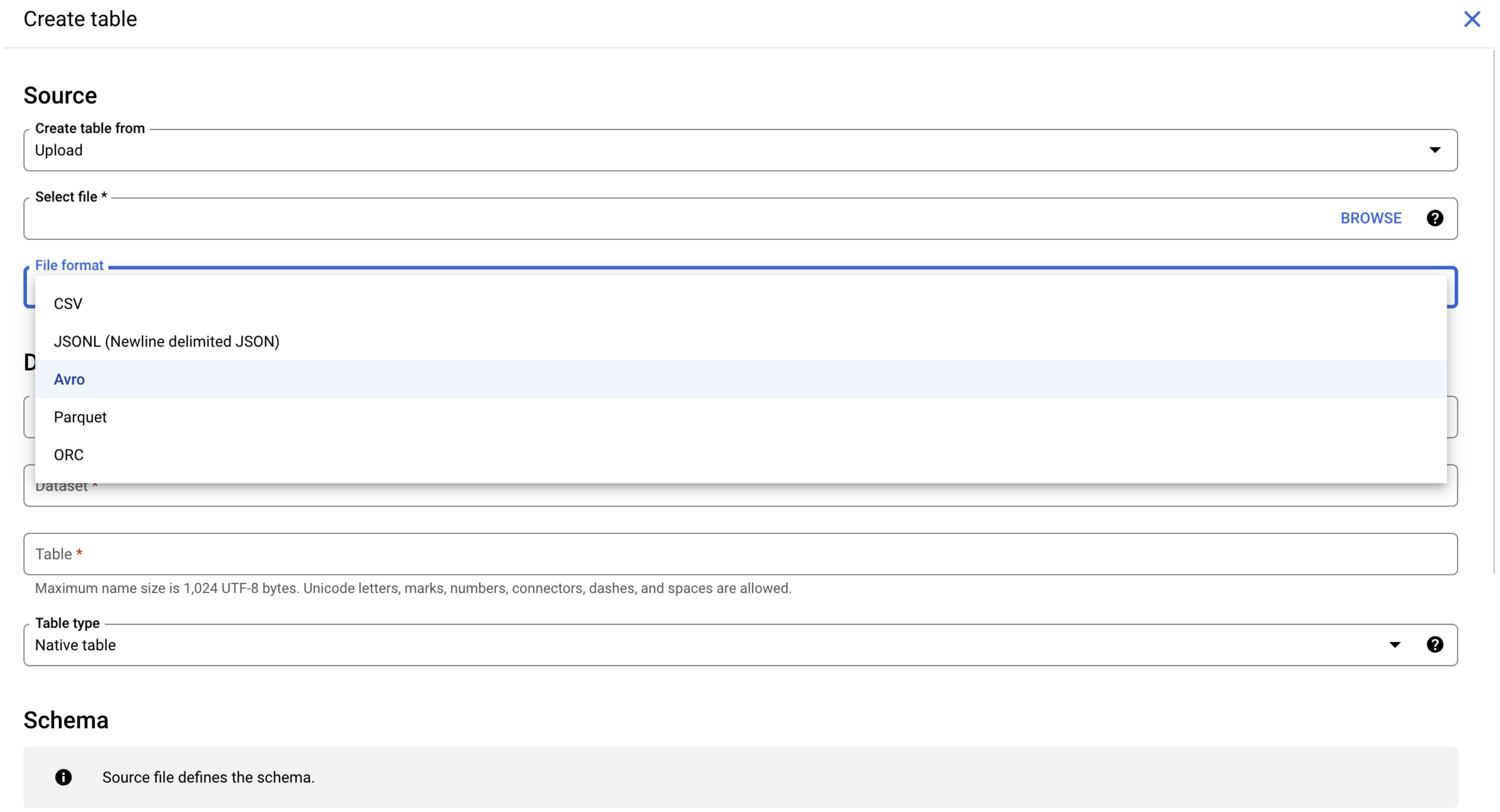
INTRODUCTION TO BIGQUERY

Matt Forrest
Field CTO

Methods of loading data

1. Loading data in the BigQuery Studio
2. Using the `bq` command line tools
3. Using the `LOAD DATA` command in SQL

Using the BigQuery Studio



Create table

Source

Create table from: Upload

Select file *

File format

CSV
JSONL (Newline delimited JSON)
Avro
Parquet
ORC

Dataset *

Table *

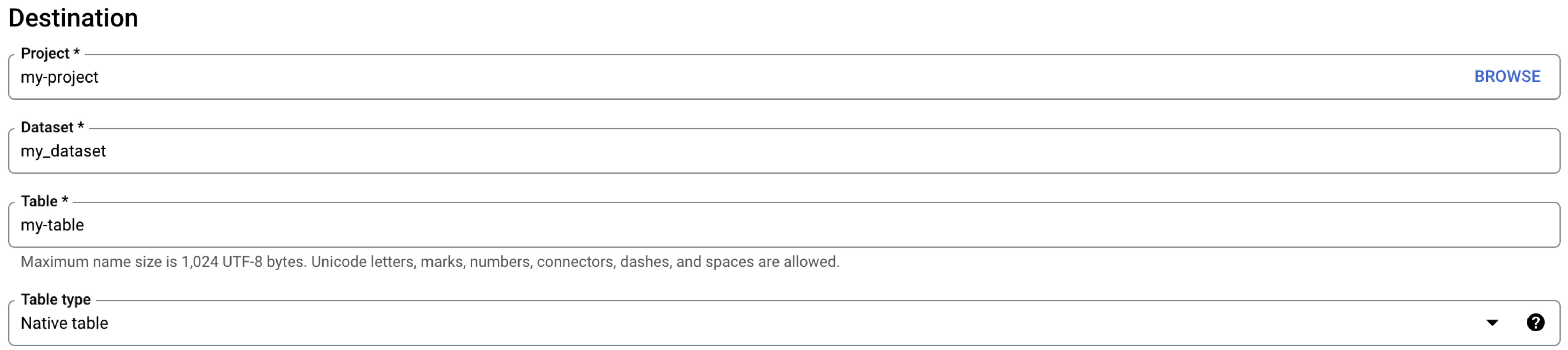
Maximum name size is 1,024 UTF-8 bytes. Unicode letters, marks, numbers, connectors, dashes, and spaces are allowed.

Table type: Native table

Schema

Source file defines the schema.

Using the BigQuery Studio

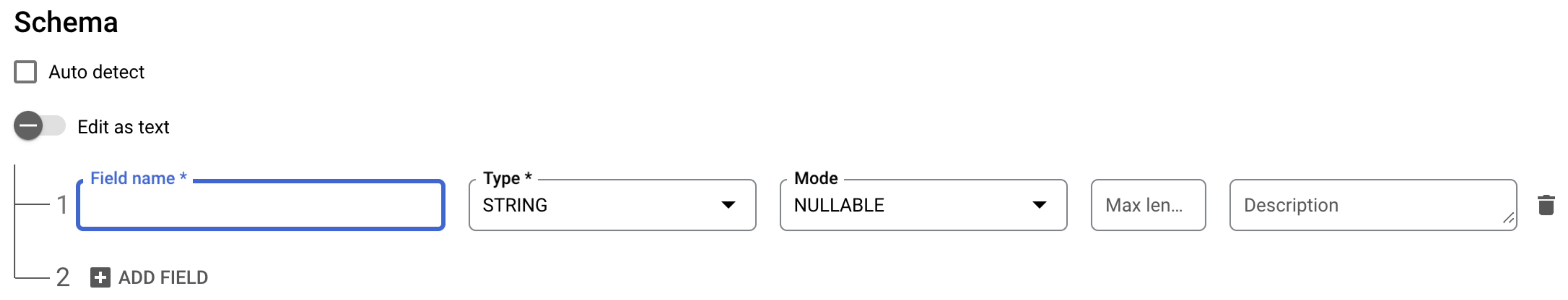


Destination

Project *	my-project
Dataset *	my_dataset
Table *	my-table
Table type	Native table

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

Using the BigQuery Studio



Schema

Auto detect

Edit as text

1 Field name * | Type * STRING | Mode NULLABLE | Max len... | Description

2 ADD FIELD

Using the bq command line tools

Example using the `bq` command line

```
bq load \
    dataset.table \
    gs://mybucket/mydata.csv \
    --source_format=CSV \
    --autodetect
```

[1] https://cloud.google.com/bigquery/docs/bq-command-line-tool#loading_data

Using LOAD DATA in SQL

An example `LOAD DATA` statement:

```
LOAD DATA INTO dataset.table
  FROM FILES(
    uris = ['gs://mybucket/mydata.csv']
    format='CSV',
    skip_leading_rows=1
  )
```

Data ingestion considerations

Using `LOAD DATA`

1. Cannot use local data.
2. Subject to load data limits.

Using `bq` and BigQuery Studio

1. Local data can be used, but files must be under 100 MB.
2. Subject to load data limits.

[1] https://cloud.google.com/bigquery/quotas#load_jobs

Let's practice!

INTRODUCTION TO BIGQUERY

Data/time types in BigQuery

INTRODUCTION TO BIGQUERY

Matt Forrest
Field CTO

Why date and times matter

1. Easily filter long datasets
2. Extract parts of dates/times
3. Partitioning strategies

Dates

Example of the DATE data type

```
SELECT
    DATE(2010, 05, 19) as launch

/*------------*
 | launch     |
 +------------+
 | 2010-05-19 |
 *------------*/
```

Timestamps

Example timestamp:

```
SELECT
    TIMESTAMP("2010-05-19 00:00:00+00") as launch

/*-------------------------*
 | launch                  |
 +-------------------------+
 | 2010-05-19 00:00:00 UTC |
 *-------------------------*/
```

Datetime and Time

Examples of time and datetime:

```
SELECT
  TIME(12, 00, 00) as noon,
  DATETIME(2019, 05, 19, 00, 00, 00) as launch

/*----------+----------------------*
 | noon     | launch               |
 +----------+----------------------+
 | 12:00:00 | 2019-05-19T00:00:00  |
 *----------+----------------------*/
```

Date and timestamp parts

Day

- `DAY` , `DAYOFWEEK` , `DAYOFYEAR` .

Week

- `WEEK` , `WEEKDAY` , `ISOWEEK` .

Month/Year

- `MONTH` , `QUARTER` , `YEAR` , `ISOYEAR` .

Time

- `HOUR` , `MINUTE` , `SECOND` , `MILLISECOND` , `MICROSECOND` .

[1] https://cloud.google.com/bigquery/docs/reference/standard-sql/timestamp_functions#timestamp_trunc

ADD, SUBTRACT, and DIFF

Adding five days

```
SELECT
    DATE_ADD(DATE '2010-05-19',
             INTERVAL 5 DAY)
             AS five_days_later;

/*---------------------*
 | five_days_later     |
 +---------------------+
 | 2010-05-24          |
 *---------------------*/
```

Finding the difference

```
SELECT
    DATE_DIFF(DATE '2010-05-24',
             DATE '2010-05-19', DAY)
             AS difference;

/*--------------*
 | difference   |
 +--------------+
 | 5            |
 *--------------*/
```

EXTRACT

Finding the day of the week

```
SELECT
    EXTRACT(DAYOFWEEK FROM DATE '2010-05-19')
             AS day_of_week;

/*---------------*
 | day_of_week  |
 +---------------+
 | 4            |
 *---------------*/
```

FORMAT

```
SELECT
    FORMAT_DATE(
      '%x', DATE '2010-05-19')
    AS with_slashes;

/*--------------*
 | with_slashes |
 +--------------+
 | 05/19/10     |
 *--------------*/
```

```
SELECT
    FORMAT_DATE(
      '%A', DATE '2010-05-19')
    AS dow;

/*--------------*
 | dow          |
 +--------------+
 | Wednesday    |
 *--------------*/
```

[1] https://cloud.google.com/bigquery/docs/reference/standard-sql/format-elements#format_elements_date_time

Current date/timestamp

Finding the current timestamp:

```
SELECT
  CURRENT_TIMESTAMP();


/*---------------------------------*
 | current_timestamp               |
 +---------------------------------+
 | 2023-11-13 17:35:25.951432 UTC  |
 *---------------------------------*/
```

Finding the current date:

```
SELECT
  CURRENT_DATE();


/*----------------*
 | current_date   |
 +----------------+
 | 2023-11-13     |
 *----------------*/
```

Cheat sheet

Data types

- Dates are exact days, timestamps are absolute dates and times

Date/timestamp parts

- Parts of a date/timestamp (e.g. `MONTH` , or `HOUR`)

Add and subtract

- Functions to add or subtract date parts (e.g. `DATE_ADD`)

Difference

- Difference between two dates by date part (e.g. `TIMESTAMP_SUB`)

Extract and format

- Extract a part of a date using date parts, format a date (e.g. `EXTRACT` and `FORMAT`)

Current date/timestamp

- Return the current date or timestamp (e.g. `CURRENT_TIMESTAMP`)

Let's practice!

INTRODUCTION TO BIGQUERY

Unstructured data

INTRODUCTION TO BIGQUERY

Matt Forrest
Field CTO

Why unstructured data is important

Example with unstructured data:

```
/*--------------+------------------------------------------*
 | customer_id  | emails                                   |
 +--------------+------------------------------------------+
 | 12345        | ['mark@google.com', 'mark@gmail.com']    |
 *--------------+------------------------------------------*/
```

ARRAYs

```
SELECT ARRAY
  (SELECT 'bigquery' UNION ALL
   SELECT 'analytics' UNION ALL
   SELECT 'sql') AS new_array;

/*-------------------------------------*
 | new_array                           |
 +-------------------------------------+
 | ['bigquery', 'analytics', 'sql']   |
 *-------------------------------------*/
```

```
SELECT ARRAY
  (SELECT 'bigquery' UNION ALL
   SELECT 'analytics' UNION ALL
   SELECT 'sql')[1] as result;

/*----------------*
 | result          |
 +----------------+
 | 'analytics'   |
 *----------------*/
```

STRUCTs

```
SELECT
  STRUCT
  <skill string, learning bool>
  ('big query', true)
  as skills;

/*--------------+--------------*
 | skills.skill | learning     |
 +--------------+------------- +
 | 'big query'  | true         |
 *--------------+--------------*/
```

```
SELECT
  STRUCT
  <skill string, learning bool>
  ('big query', true).skill
  as key;

/*---------------+
 | key.          |
 +---------------+
 | 'big query'   |
 *---------------*/
```

ARRAY_LENGTH and ARRAY_CONCAT

```
SELECT
  ARRAY_LENGTH(
    [
      'mark@google.com',
      'mark@gmail.com'
    ]) as len;

/*-------*
 | len   |
 +-------+
 | 2     |
 *-------*/
```

```
SELECT
  ARRAY_CONCAT(
    ['one'], ['two']) as new_array;

/*--------------------*
 | new_array          |
 +--------------------+
 | ['one', 'two']     |
 *--------------------*/
```

UNNEST

Unnest-ing our email data:

```
SELECT
    *
FROM
    UNNEST(['mark@google.com', 'mark@gmail.com']) as emails;

/*--------------------+
 | emails             |
 +--------------------+
 | 'mark@google.com'  |
 | 'mark@gmail.com'   |
 *--------------------*/
```

UNNEST with STRUCTs

Example data - STRUCT inside an ARRAY:

```
[
    {'big query': true},
    {'sql': true}
]
```

```
SELECT
    my_skills.skill,
    my_skills.learned
FROM
    UNNEST(skills) as my_skills
/*--------------+--------------*
 | skill        | learned      |
 +--------------+------------- +
 | 'big query'  | true         |
 +--------------+------------- +
 | 'sql'        | true         |
 *--------------+--------------*/
```

SEARCH

Using search with email data:

```
SELECT
    SEARCH(['mark@google.com', 'mark@gmail.com'], 'gmail.com') as results;

/*----------*
 | results  |
 +----------+
 | true     |
 *----------*/
```

Unstructured data cheat sheet

ARRAYs

- Similar to lists with ordered values
- Values can accessed via a base 0 index (e.g., `my_array[0]`)

STRUCTs

- Similar to JSON or a dictionary
- Can have any structure with multiple data types
- Structure must be the same for all rows in that column

ARRAY_CONCAT/ARRAY_LENGTH

- Concatenate two or more arrays and measure an array length

UNNEST

- Allows you to flatten ARRAY data

SEARCH

- Search across ARRAYs or STRUCTs to find matching values

Let's practice!