

Google Cloud Platform

Clauses and Functions

BigQuery for Data Analysts

V1.2

Approximate timing: 90 minutes

Agenda

- 1 BigQuery SQL
- 2 Standard SQL Support
- 3 BigQuery Functions
- 4 User-Defined Functions
- 5 Query Recipes
- 6 Quiz & Lab

BigQuery SQL (Legacy SQL)

- BigQuery SQL is very similar to ANSI SQL extensions
 - Optimized for big data analytics
 - Adds many powerful functions
- BigQuery provides common, familiar clauses and functions in a slightly different dialect from traditional SQL
 - SELECT ... FROM - Sub-select
 - SELECT ... FROM - Joins
 - SELECT ... FROM - Sub-select Joins
 - SELECT ... FROM - Union
- [Additional information on query syntax](#)
- Currently still the default query language (can be deselected)

BigQuery Data Types - Legacy SQL

Data type	Possible value
STRING	UTF-8 encoded string -- 2 MB max
INTEGER	64-bit signed integer
FLOAT	Double-precision <u>floating-point</u> format
BOOLEAN	<ul style="list-style-type: none">● CSV format: true or false (case insensitive), or 1 or 0● JSON format: true or false (case insensitive)
RECORD	A collection of one or more fields
TIMESTAMP	TIMESTAMP data types can be described in two ways: UNIX timestamps or calendar date times. BigQuery stores TIMESTAMP data internally as epoch with microsecond precision.

Notes:

You do not have to specify length for strings. For more information on all data types supported by legacy SQL, see:

<https://cloud.google.com/bigquery/data-types>.

SQL versus BigQuery SQL - Filtering

SQL - LIKE

```
SELECT LOWER(word) AS word,  
       word_count AS frequency,  
       corpus  
FROM Shakespeare  
WHERE corpus LIKE '%king%'  
AND  
       LENGTH(word) > 5  
ORDER BY frequency DESC
```

BigQuery SQL - CONTAINS

```
SELECT LOWER(word) AS word,  
       word_count AS frequency,  
       corpus  
FROM  
[publicdata:samples.Shakespeare]  
WHERE corpus CONTAINS 'king'  
AND  
       LENGTH(word) > 5  
ORDER BY frequency DESC
```

Notes:

Comparing traditional SQL and BigQuery SQL – differences highlighted -
Contains operator vs. like operator

SQL versus BigQuery SQL - JOINS

SQL

```
SELECT emp.Name AS  
EMPLOYEE,  
dep.Name AS DEPARTMENT  
FROM EMPLOYEE emp,  
DEPARTMENT dep  
WHERE emp.dept_id =  
dep.dept_id  
ORDER BY emp.NAME
```

BigQuery SQL

```
SELECT emp.Name AS EMPLOYEE,  
dep.Name AS DEPARTMENT  
FROM [myproject:mycompany.EMPLOYEE]  
AS emp  
JOIN  
[myproject:mycompany.DEPARTMENT] AS  
dep  
ON emp.dept_id = dep.dept_id  
ORDER BY EMPLOYEE
```

Note: The structure of this query is similar to ANSI SQL syntax except the table references.

Notes:

Joins in BigQuery use the JOIN syntax. This is due to the fact that joins were not initially supported in BigQuery but was added later. BigQuery supports INNER JOINS, OUTER JOINS, and CROSS JOINS. In traditional SQL you can do "on emp.dept_id = dep.dept_id", so the deviation is very minimal here.

SQL versus BigQuery SQL - UNION

SQL

```
SELECT DISTINCT
dept_id
FROM (SELECT dept_id
FROM EMPLOYEE
UNION ALL
SELECT dept_id FROM
DEPARTMENT)
```

BigQuery SQL

```
SELECT dept_id
FROM (SELECT dept_id FROM
mycompany.EMPLOYEE),
(SELECT dept_id FROM
mycompany.DEPARTMENT)
GROUP BY dept_id
```

Notes:

Unions in BigQuery SQL are comma separated lists of tables. Students will see later in the course where this syntax will be advantageous for analytics.

Notice that BigQuery SQL does not support a DISTINCT statement. You must use a group by to get unique rows.

BigQuery SQL - SELECT FROM JOIN

- A BigQuery JOIN takes two tables and matches every row in the first table against every row in the second table
- The **ON** clause is required (except for CROSS JOIN)
- BigQuery only supports equijoins
- BigQuery supports **INNER**, **[FULL|RIGHT|LEFT] OUTER** and **CROSS JOIN** operations
 - The default is **INNER**

BigQuery SQL

```
SELECT ...
FROM DATASET.TABLE1
  [INNER|[FULL|RIGHT|LEFT]
  OUTER|CROSS]
JOIN
  DATASET.TABLE2
  [ON join_condition_1 [...
  AND join_condition_n]
```

BigQuery SQL - SELECT FROM Union

- Unlike many other SQL-based systems, BigQuery uses the comma syntax to indicate table unions, not joins
- Query over several tables with the same schema

BigQuery SQL

```
SELECT ...  
FROM  
    [dataset.table1],[dataset.  
table2],[dataset.table3] AS  
T1  
JOIN  
    (SELECT ... FROM  
dataset.table4) as T4 ON T1.y  
= T4.y  
WHERE ...
```

BigQuery SQL - Sub-SELECT

- Some complex processing is easier to express with a sub-select statement
- The result of the sub-SELECT clause appears as a source table to the outer SQL selection
- Sub-SELECT result must be <128MB compressed

```
SELECT ...  
FROM (  
  SELECT ...  
  FROM ...  
)
```

BigQuery SQL

Notes:

Sub-select clauses can greatly simplify the conceptual structure of a SQL query. The sub-select is required to be within the 128M result set limit.

BigQuery SQL - Sub-SELECT JOIN

- Create compound and complex queries using sub-SELECT in JOINS
- Simplifies overall query structure

```
BigQuery SQL
SELECT
  a.title AS title,
  edit_count,
  MAX(num_characters) AS max_entry_size
FROM
  [publicdata:samples.wikipedia] b
JOIN (
  SELECT
    TOP(title, 100) AS title,
    COUNT(*) AS edit_count
  FROM
    [publicdata:samples.wikipedia]) a
ON
  a.title = b.title
GROUP BY
  title,
  edit_count
```

Notes:

BigQuery encourages the use of taking simple queries to make compound queries. Many DBMS frown on this practice due to overhead involved to execute. However BigQuery's parallel architecture performs better using this approach.

Agenda

- 1 BigQuery SQL
- 2 Standard SQL Support
- 3 BigQuery Functions
- 4 User-Defined Functions
- 5 Query Recipes
- 6 Quiz & Lab

Standard SQL **Beta** (1 of 2)

- SQL 2011 compliant SQL new to BigQuery
- Removes workarounds in BigQuery SQL
 - Example: Comma is now JOIN, not UNION ALL
 - But does include extensions for nested and repeated data
- Port SQL workloads directly into BigQuery without translation

Notes:

For more information on standard SQL support in BigQuery, see:

<https://cloud.google.com/bigquery/sql-reference/>.

Standard SQL **Beta** (2 of 2)

- For existing BigQuery users, new data types and features
 - Richer data types: STRUCTs, ARRAYs, DATEs, TIMEs
 - Easier to operate on or generate structured tables
 - Additional support for timestamps
 - Support for complex JOIN predicates
 - Named subqueries using WITH clauses
 - Subqueries can go almost anywhere (SELECT, WHERE, ...)
 - Support for EXISTS subquery
 - COUNT(DISTINCT) is now exact and scalable
 - JOIN conditions allow non-equality comparisons between keys (Theta JOIN)

BigQuery Data Types - Standard SQL

Data type	Possible value
STRING	Variable-length character (Unicode) data
INT64	64-bit integer
FLOAT64	Double-precision (approximate) decimal values
BOOL	True or false (case insensitive)
ARRAY	Ordered list of zero or more elements of any non-ARRAY type.
STRUCT	Container of ordered fields each with a type (required) and field name (optional).
TIMESTAMP	Represents an absolute point in time, with precision up to microseconds. Values range between the years 1 and 9999, inclusive.

Notes:

For more information on all data types supported by standard SQL, see: <https://cloud.google.com/bigquery/sql-reference/data-types>.

COUNT(DISTINCT) Example

```
#Rainiest areas of Washington state in 2015
WITH WashingtonStations AS (
  SELECT DISTINCT name, weather.stn station_id
  FROM 'bigquery-public-data.noaa_gsod.stations' station
  INNER JOIN 'bigquery-public-data.noaa_gsod.gsod2015' weather
  ON station.usaf = weather.stn
  WHERE state = 'WA')
SELECT name,
  (SELECT COUNT(*)
   FROM 'bigquery-public-data.noaa_gsod.gsod2015' inner
   WHERE outer.station_id = inner.stn) AS rainy_days
FROM WashingtonStations outer
ORDER BY reported_rainy_days DESC LIMIT 5;
```

ARRAY/STRUCT Example

```
#Top two Hacker News articles by day
WITH TitlesAndScores AS (
  SELECT
    ARRAY_AGG(STRUCT(title, score)) AS titles,
    EXTRACT(DATE FROM time_ts) AS date
  FROM 'bigquery-public-data.hacker_news.stories'
  WHERE score IS NOT NULL AND title IS NOT NULL
  GROUP BY date)
SELECT date,
  ARRAY(SELECT AS STRUCT title, score
        FROM UNNEST(titles) ORDER BY score DESC LIMIT 2)
  AS top_articles
FROM TitlesAndScores;
```

Notes:

With standard SQL, you can now store data in arrays natively. In BigQuery SQL, you had to store a list of values in a record and then use string concatenation.

You can also store STRUCTs natively. Previously, you had to store two pieces of data as one record and then use string concatenation.

JOIN Condition Example

```
#Top name frequency in Shakespeare
WITH TopNames AS (
  SELECT name, SUM(number) AS occurrences
  FROM 'bigquery-public-data.usa_names.usa_1910_2013'
  GROUP BY name
  ORDER BY occurrences DESC LIMIT 100)
SELECT name, SUM(word_count) AS frequency
FROM TopNames
JOIN 'bigquery-public-data.samples.shakespeare'
ON STARTS_WITH(word, name)
GROUP BY name
ORDER BY frequency DESC LIMIT 10;
```

Notes:

In this example, the JOIN condition uses a non-equality comparison between keys.

Agenda

- 1 BigQuery SQL
- 2 Standard SQL Support
- 3 BigQuery Functions
- 4 User-Defined Functions
- 5 Query Recipes
- 6 Quiz & Lab

BigQuery Functions

- BigQuery provides many advanced functions including:
 - Regular expression functions
 - Window functions
 - Date/time functions
 - Casting
 - [And many others](#)

Notes:

For more information on BigQuery functions in legacy SQL, see:
<https://cloud.google.com/bigquery/query-reference>.

Regular Expression Functions (1 of 2)

- Legacy SQL supports REGEXP_MATCH, REGEXP_EXTRACT and REGEXP_REPLACE
- Standard SQL supports REGEXP_CONTAINS, REGEXP_EXTRACT, REGEXP_EXTRACT_ALL, and REGEXP_REPLACE
- Enables analysis of unstructured data (free text comments, queries, and so on)
- Negligible performance impact

Notes:

In legacy SQL, REGEXP_MATCH returns true if str matches the regular expression. For string matching without regular expressions, use CONTAINS instead. Standard SQL supports REGEXP_CONTAINS.

Regular Expression Functions (2 of 2)

REGEX example - Legacy SQL

```
SELECT word, COUNT(word) as
count
FROM
publicdata:samples.shakespear
e
WHERE
(REGEXP_MATCH(word,r'^\w\w'\w\w')
)
GROUP BY word
ORDER BY count DESC
LIMIT 3;
```

Query Results 12:07pm, 17 Dec 2013

Row	word	count	
1	ne'er	42	
2	we'll	35	
3	We'll	33	

Notes:

This is a query that matches any five letter words with an apostrophe in the middle. This query demonstrated the flexibility of BigQuery. You don't need to pre-index the table in order for the search to return quickly. MySQL supports regexp as well. Here is an example - Select name from world.City where name REGEXP '^.*burg.*\$'. But it only supports REGEXP_MATCH. Oracle supports SELECT first_name, last_name FROM employees WHERE REGEXP_LIKE (first_name, '^Ste(v|ph)en\$');

Window (Analytic) Functions (1 of 2)

- Window functions enable calculations on specific partition, or "window", of result set
- Each window function expects an **OVER** clause that specifies the partition
- The **PARTITION BY** clause (optional) allows window functions to partition data and parallelize execution
- The **ORDER BY** clause (optional) defines order within each partition
 - If no ORDER BY present, row ordering within partition is non-deterministic

Notes:

Enable calculations on a specific partition, or "window", of a result set. Each window function expects an OVER clause that specifies the partition.

Window (Analytic) Functions (2 of 2)

- Multiple Window functions available including:
 - `CUME_DIST()` -- Returns cumulative distribution of a value in a group
 - `DENSE_RANK()` -- Returns integer rank of a value in a group
 - `ROW_NUMBER()` -- Returns current row number of the query result
 - `RANK()` -- Returns the integer rank of a value in a group of values
 - `PERCENT_RANK()` -- Returns the rank of the current row, relative to the other rows in the partition

Window Function Examples (1 of 2)

Window function using RANK

```
SELECT
  corpus,
  word,
  word_count,
  RANK() OVER (PARTITION BY
  corpus ORDER BY word_count DESC)
  rank,
FROM
  [publicdata:samples.shakespeare]
WHERE
  length(word) > 10 AND
  word_count > 10
LIMIT 40
```

Row	corpus	word	word_count	rank
1	kinghenryv	WESTMORELAND	15	1
2	kinghenryviii	Chamberlain	53	1
3	3kinghenryvi	NORTHUMBERLAND	21	1
4	3kinghenryvi	Plantagenet	14	2
5	othello	handkerchief	29	1
6	1kinghenryiv	WESTMORELAND	16	1
7	1kinghenryiv	NORTHUMBERLAND	13	2
8	1kinghenryiv	Westmoreland	12	3
9	1kinghenryvi	PLANTAGENET	32	1
10	1kinghenryvi	Plantagenet	12	2

Notes:

The sample query select over Shakespeare's work and find all the words that are longer than 10 with a minimum of 10 word counts, and rank by word count.

Date and Time Functions

- Enable date and time manipulation for timestamps, date strings and TIMESTAMP data types
- BigQuery uses Epoch time
- Return values based on the UTC time zone (default)
- Standard SQL has many date/time functions, including:
 - DATE(year, month, day): Constructs DATE from INT64 values representing year, month, and day
 - DATE (timestamp): Converts timestamp_expression to DATE and supports timezone parameter (optional)
 - DATE_DIFF(date_expression, date_expression, date_part): Computes number of specified date_part differences between two date expressions

Notes:

Date/timestamp parsing and conversion functions in legacy SQL include:

- NOW() function: returns microseconds elapsed from 1/1/1970 (Epoch)
- CURRENT_TIMESTAMP() returns YYYY-MM-DD HH:MM:SS
- CURRENT_DATE() returns YYYY-MM-DD
- CURRENT_TIME() returns HH:MM:SS

Legacy SQL doesn't support the TRUNCATE function as in other systems; instead, use CURRENT DATE to get the date without the time. Standard SQL supports the DATE function which constructs a DATE from INT64 values representing the year, month, and day.

Casting

- Change the datatype of a numeric expression
- Ensures arguments in a comparison function have the same data type
- Explicit in legacy SQL (Implied casting is not supported)
 - Standard SQL supports coercion (implicit conversion)
- Example - Convert a float to a string without scientific notation:

Casting example - Legacy SQL

```
SELECT STRING(INTEGER(f)) + '.' +  
SUBSTR(String(f-INTEGER(f)), 3)  
FROM (SELECT 5302014.5642 f)
```

Query Results Oct 27, 2015, 3:31:13 PM

Table		JSON
Row	f0_	
1	5302014.564200	

Agenda

- 1 BigQuery SQL
- 2 Standard SQL Support
- 3 BigQuery Functions
- 4 User-Defined Functions
- 5 Query Recipes
- 6 Quiz & Lab

User-Defined Functions (UDFs)

- Allow SQL queries to use programming logic
- Allow additional functionality not supported by standard SQL
 - Loops, complex conditions, non-trivial string parsing
- Functions are written in JavaScript
- Similar to map functions in MapReduce
- Output can be any BigQuery-supported data type

Notes:

User defined functions allow BigQuery to execute code snippets against data.

UDF Structure

- Two formal parameters: ROW and EMIT
- ROW
 - An input row
- EMIT
 - A hook to JavaScript object to collect output
 - Object represents a single row of output
 - Can be called more than once

UDF

```
Function  
name(row,emit)  
{  
  emit(<outputdata>);  
}
```

UDF Registration

- UDF name must be registered so BigQuery can invoke it
- Registration name and function name may be different
- Input column names must match table column names
- Output schema must be in JSON format

```
bigquery.defineFunction(  
  '<UDF name>', // Name used to  
                // call the function from SQL  
  ['<col1>', '<col2>'], // Input  
                             // column names  
                             // JSON representation of the  
                             // output schema  
  [<output schema>],  
  // UDF definition or reference  
  <UDF definition or reference>  
);
```

UDF

UDF Example

UDF

```
// UDF definition
function urlDecode(row, emit) {emit({title: decodeHelper(row.title),
    requests: row.num_requests});
}
// Helper function with error handling
function decodeHelper(s) {
  try {return decodeURI(s);
  } catch (ex) {return s;}
}
// UDF registration
bigquery.defineFunction( 'urlDecode', // Name used to call the function from SQL
  ['title', 'num_requests'], // Input column names
  // JSON representation of the output schema
  [{name: 'title', type: 'string'},
   {name: 'requests', type: 'integer'}],

  urlDecode // The function reference
);
```

UDF Query Example

UDF query

```
SELECT requests, title
FROM
  urlDecode(
    SELECT
      title, sum(requests) AS num_requests
    FROM
      [fh-bigquery:wikipedia.pagecounts_201504]
    WHERE language = 'fr'
    GROUP BY title
  )
WHERE title LIKE '%ç%'
ORDER BY requests DESC
LIMIT 100
```

Other UDF Details

- UDF failure causes calling query failure
- UDF error handling supported with helper functions
- Use web UI to add UDFs when running queries
- UDFs can be stored and called from cloud storage

UDF Constraints

- Max output data size: 5 MB per input row
- Max number of UDFs that can run concurrently per user = 3
- Native code JavaScript functions are not supported
- The DOM objects **Window**, **Document** and **Node**, and functions that require them, are unsupported
- Bitwise operations in JavaScript handle only the most significant 32 bits

UDF Best Practices

- Use UDF test tool to debug to avoid incurring BigQuery charges
- Avoid persistent mutable state
 - Do not store or access mutable state in UDF calls
 - Each BQ node has local javascript processing environment that may produce local values
- Use memory efficiently to avoid memory exhaustion on local JavaScript environments
- Explicitly list SELECT columns instead of SELECT * (not supported)
- Pre-filter input to limit the size of data on which UDF will run

Notes:

Use of the UDF test tool does not incur billing charges. Since BigQuery shards your query across many nodes, each node has a standalone processing environment and memory values are not shared. The JavaScript processing environment has limited memory available per query UDF queries that accumulate too much local state may fail. You must explicitly list the columns being selected from a UDF - SELECT * FROM <UDF name> is not supported. If your input can be easily filtered down before being passed to a UDF, it will likely run faster and cheaper.

Agenda

- 1 BigQuery SQL
- 2 Standard SQL Support
- 3 BigQuery Functions
- 4 User-Defined Functions
- 5 Query Recipes
- 6 Quiz & Lab

BigQuery Recipes

- Examples of queries for reporting applications
- Highlights some functions available
- Code comments explain purpose
- See [BigQuery documentation](#) for detailed list of functions and operators

Notes:

The following slides include query recipe examples showing how to do some common reporting applications.

Pivot

Pivot

```
/* Find 100 largest words in Shakespeare's works and display the number
of
occurrences of those words in four of Shakespears more popular works. */
SELECT word,
SUM(IF(corpus = 'kinglear', corpus_total, 0)) AS kinglear,
SUM(IF(corpus = 'tempest', corpus_total, 0)) AS tempest,
SUM(IF(corpus = 'macbeth', corpus_total, 0)) AS macbeth,
SUM(IF(corpus = 'hamlet', corpus_total, 0)) AS hamlet,
SUM(corpus_total) AS [total]
FROM (SELECT word,LENGTH(word) AS word_len, corpus, SUM(word_count) AS
corpus_total
FROM [publicdata:samples.shakespeare]
WHERE LENGTH(word) > 10
GROUP BY word, word_len, corpus)
GROUP BY word, word_len
ORDER BY word_len DESC
LIMIT 100
```

Notes:

Since BigQuery does not have a Pivot function, users need to build their pivot routine using conditional logic after the SELECT statement. Look in the example to see how the IF condition is being used to create the pivot.

Cohort Analysis

Cohort

```
/* Counts of Wikipedia contributors who contribute only to pages on Manning brothers vs those who
contributed to Manning brothers pages and other pages. Query highlights EVERY and SOME functions. */
SELECT peyton_all, peyton_some, eli_all, eli_some, COUNT(1) AS NUM, AVG(edits) AS
avg_edits
FROM (
  SELECT contributor_id,
    EVERY(peyton_edit) AS peyton_all, SOME(peyton_edit) AS peyton_some,
    EVERY(eli_edit) AS eli_all, SOME(eli_edit) AS eli_some,
    COUNT(1) AS edits
  FROM (
    SELECT contributor_id,
      (LOWER(title) CONTAINS 'peyton manning') AS peyton_edit,
      (LOWER(title) CONTAINS 'eli manning') AS eli_edit
    FROM [publicdata:samples.wikipedia])
  GROUP BY contributor_id
  HAVING peyton_all OR peyton_some OR eli_all OR eli_some)
GROUP BY peyton_all,peyton_some,eli_all,eli_some
ORDER BY peyton_all,peyton_some,eli_all,eli_some
```

Notes:

When trying to make sense of large data sets, it is common to identify cohorts of sets of entities that have a specific property. This example shows the Wikipedia contributors that edited a particular title.

Take note the use of the EVERY and SOME functions. EVERY returns true if true for all inputs. SOME returns true if true for some inputs. The example shows the number of contributors that contributed to only the title articles or if they contributed to other articles.

Trailing Averages

Trailing avg

```
/* Demonstrates windowing functions to calculate moving average on number
 * of user activities in github. Outermost query combines trailing values into a
 * weighted average paying attention to missing values. */
SELECT start_date,
  ((num_0 + if(num_1 > -1, num_1, num_0) * 0.5 + if(num_2 > -1, num_2, num_0) * 0.25)
 / 1.75) AS smooth_num
FROM (
  SELECT start_date, num_0,
    LAG(num_0, 1, integer(-1))
      OVER (ORDER BY start_date) AS num_1,
    LAG(num_0, 2, INTEGER(-1))
      OVER (ORDER BY start_date) AS num_2
  FROM (
    SELECT DATE(created_at) as start_date,
      INTEGER(COUNT(1)) num_0
    FROM [publicdata:samples.github_timeline]
    GROUP BY start_date))
  ORDER BY smooth_num desc
```

Notes:

Trailing or Moving Averages are often used in graphing routines. This query demonstrates the use of windowing functions across the data. The LAG function is used get offset values to compute the weighted average in the outer query.

Agenda

- 1 BigQuery SQL
- 2 Standard SQL Support
- 3 BigQuery Functions
- 4 User-Defined Functions
- 5 Query Recipes
- 6 Quiz & Lab

Module Review (1 of 2)

Which two of the following are true?

- Implied casting is supported in BigQuery standard SQL
- The registration name and function name must be the same for a user-defined function
- The DATE function in returns data in YYYY-MM-DD format
- Native code JavaScript functions are supported as user-defined functions

Module Review (2 of 2)

Which of the following statements are true?

(select 2 of the available options)

- There is a 10Mb limit on the amount of data that a UDF can output on a row
- UDFs are written using JavaScript
- The maximum data size for a federated table in a query is 10 MB
- In order to create a federated table, you need to provide the table definition and the URI path to Google Cloud Storage

Lab

Use advanced query clauses and functions, including: User-defined functions (UDFs), wildcard functions, and window functions

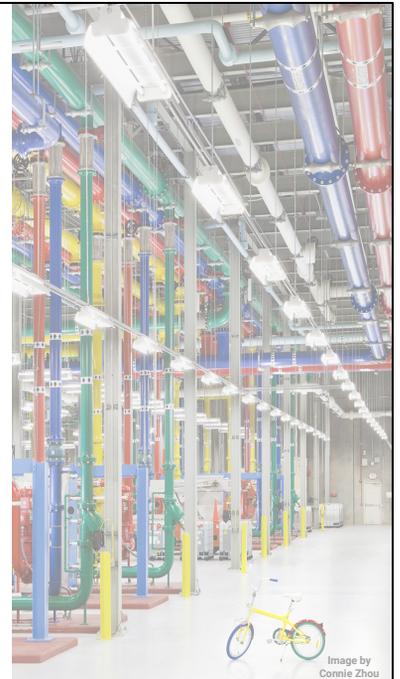


Image by
Cennie Zhou

Resources

- Querying data
<https://cloud.google.com/bigquery/querying-data>
- UDF test
tool
<http://storage.googleapis.com/bigquery-udf-test-tool/testtool.html>
- Legacy SQL query
reference
<https://cloud.google.com/bigquery/query-reference>
- Standard SQL query
reference
<https://cloud.google.com/bigquery/sql-reference/>

Module Review Answers (1 of 2)

Which two of the following are true?

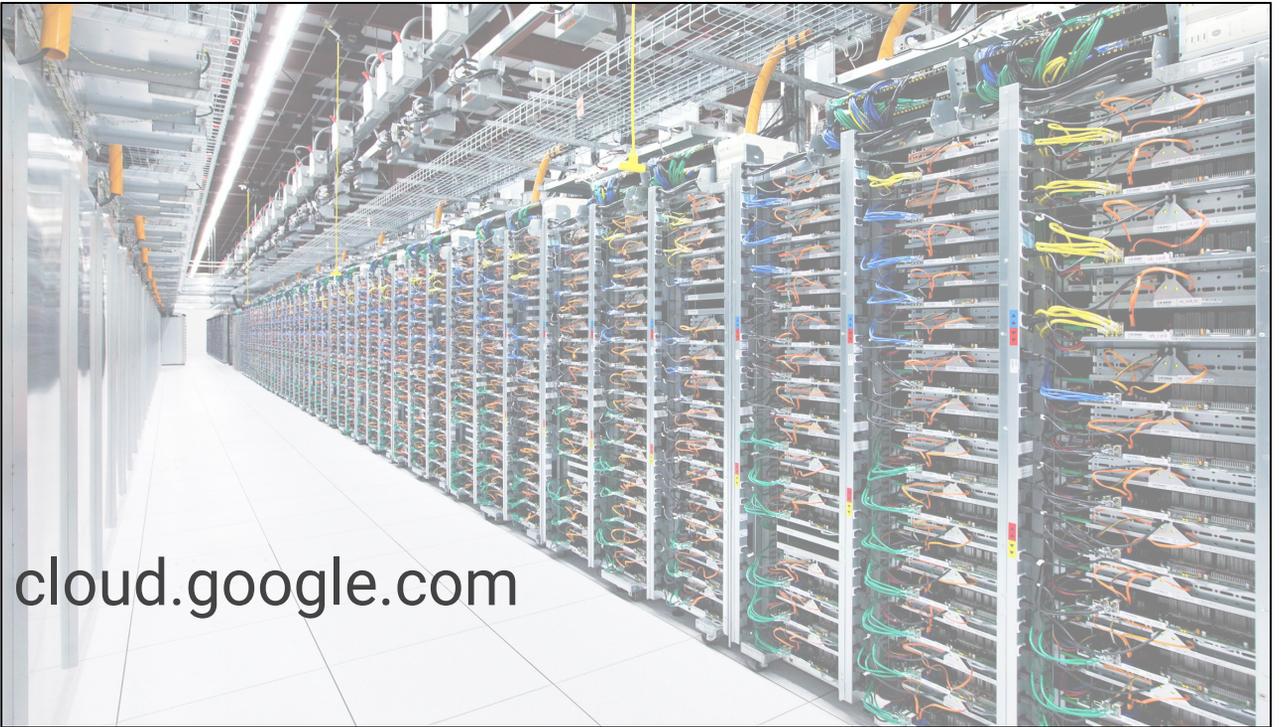
- ✓ Implied casting is supported in BigQuery standard SQL
- The registration name and function name must be the same for a user-defined function
- ✓ The DATE function in returns data in YYYY-MM-DD format
- Native code JavaScript functions are supported as user-defined functions

Module Review Answers (2 of 2)

Which of the following statements are true?

(select 2 of the available options)

- There is a 10Mb limit on the amount of data that a UDF can output on a row
- ✓ UDFs are written using JavaScript
- The maximum data size for a federated table in a query is 10 MB
- ✓ In order to create a federated table, you need to provide the table definition and the URI path to Google Cloud Storage



cloud.google.com