Module 10

Advanced Functions and Clauses

In this module we will:

- **Introduce Advanced Functions (Statistical, Analytic, User-Defined)**
- Discuss Effective Sub-query and CTE design

Google Cloud

Use the Right Function for the Right Job

- String Manipulation Functions - FORMAT( )
- Aggregation Functions - SUM( ) COUNT( ) AVG( ) MAX( )
- Data Type Conversion Functions - CAST( )
- Date Functions - PARSE_DATETIME( )
- **Statistical Functions**
- **Analytic Functions**
- **User-defined Functions**

BigQuery Functions Reference

Google Cloud

Run Statistical Functions over Values

```
SELECT
  STDDEV(noemplyeesw3cnt) AS st_dev_employee_count,
  CORR(totprgmrevnue, totfuncexpns) AS corr_rev_expenses
FROM
  `bigquery-public-data.irs_990.irs_990_2015`
```

How **correlated** do you think Program Revenue and Total Functional Expenses are?

More SQL Statistical Functions

Google Cloud

Run Statistical Functions over Values

```
SELECT
  STDDEV(noemplyeesw3cnt) AS st_dev_employee_count,
  CORR(totprgmrevnue, totfuncexpns) AS corr_rev_expenses
FROM
  `bigquery-public-data.irs_990.irs_990_2015`
```

Row	st_dev_employee_count	corr_rev_expenses
1	1579.8005361247351	0.9761801901905149

[More SQL Statistical Functions]

Google Cloud

Try **Approximate Aggregate Functions** when Close Enough will do

```
#standardSQL
SELECT
  APPROX_COUNT_DISTINCT(ein) AS approx_count,
  COUNT(DISTINCT ein) AS exact_count
FROM
  `bigquery-public-data.irs_990.irs_990_2015`
```

Row	approx_count	exact_count
1	276880	275077

Table JSON

More SQL Approximation Functions

Approximate Users Per Year of All Github User Logins

```
#standardSQL
SELECT
   CONCAT('20', _TABLE_SUFFIX) year,
   APPROX_COUNT_DISTINCT(actor.login) approx_cnt
FROM `githubarchive.year.20*`
GROUP BY year
ORDER BY year

# 3.8s elapsed, 8.37 GB processed
```

Row	year	approx_cnt
1	2011	540440
2	2012	1188211
3	2013	2208240
4	2014	3117587
5	2015	4440679
6	2016	6643627

[Example from Google Big Data Blog]()

Bonus: Approximate Unique Github Users Since 2011

```
#standardSQL
WITH github_year_sketches AS (
SELECT
    CONCAT('20', _TABLE_SUFFIX) AS year,
    APPROX_COUNT_DISTINCT(actor.login) AS approx_cnt,
    HLL_COUNT.INIT(actor.login) AS sketch # HyperLogLog Estimation
FROM `githubarchive.year.20*`
GROUP BY year
ORDER BY year)

SELECT HLL_COUNT.MERGE(sketch) AS approx_unique_users
FROM `github_year_sketches`
#4.2s elapsed, 8.37 GB processed
#11,334,294 Unique Github Users, Only 0.3% off exact count
```

← we'll cover WITH clauses shortly

[Example from Google Big Data Blog]

Use **Analytic Window Functions** for Advanced Analysis

- Standard aggregations
  - `SUM`, `AVG`, `MIN`, `MAX`, `COUNT`, `etc.`
- Navigation functions
  - `LEAD()` – Returns the value of a row *n* rows ahead of the current row
  - `LAG()` – Returns the value of a row *n* rows behind the current row
  - `NTH_VALUE()` – Returns the value of the *n*th value in the window
- Ranking and numbering functions
  - `CUME_DIST()` – Returns the cumulative distribution of a value in a group
  - `DENSE_RANK()` – Returns the integer rank of a value in a group
  - `ROW_NUMBER()` – Returns the 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

Example: **RANK() Function** for Aggregating over Groups of Rows

firstname	department	startdate
Andrew	1	1/23/1999
Jacob	1	7/11/1990
Daniel	2	6/24/2004
Anna	1	10/7/2001
Pierre	1	2/22/2009
Ruth	2	6/6/1998
Anthony	1	11/29/1995
Isabella	2	9/28/1997
Jose	2	3/17/2013

`PARTITION BY department`

firstname	department	startdate
Andrew	1	1/23/1999
Jacob	1	7/11/1990
Anna	1	10/7/2001
Pierre	1	2/22/2009
Anthony	1	11/29/1995

firstname	department	startdate
Ruth	2	6/6/1998
Daniel	2	6/24/2004
Jose	2	3/17/2013
Isabella	2	9/28/1997

`ORDER BY startdate`

firstname	department	startdate
Jacob	1	7/11/1990
Anthony	1	11/29/1995
Andrew	1	1/23/1999
Anna	1	10/7/2001
Pierre	1	2/22/2009

firstname	department	startdate
Isabella	2	9/28/1997
Daniel	2	6/24/2004
Jose	2	3/17/2013
Ruth	2	6/6/2013

`RANK()`

firstname	department	startdate	rank
Jacob	1	7/11/1990	1
Anthony	1	11/29/1995	2
Andrew	1	1/23/1999	3
Anna	1	10/7/2001	4
Pierre	1	2/22/2009	5

firstname	department	startdate	rank
Isabella	2	9/28/1997	1
Daniel	2	6/24/2004	2
Jose	2	3/17/2013	3
Ruth	2	6/6/2013	4

Get the oldest ranking employee ***by each department***

Sometimes called a "window" function

More SQL Analytic Functions

Example: **RANK() Function** for Aggregating over Groups of Rows

```
SELECT firstname, department, startdate,
  RANK() OVER ( PARTITION BY department ORDER BY startdate ) AS rank
FROM Employees;
```


Google Cloud

Components of a **User-Defined Function (UDF)**

- CREATE TEMPORARY FUNCTION. Creates a new function. A function can contain zero or more named_parameters
- RETURNS [data_type]. Specifies the data type that the function returns.
- Language [language]. Specifies the language for the function.
- AS [external_code]. Specifies the code that the function runs.

```
CREATE TEMPORARY FUNCTION greeting(a STRING)
RETURNS STRING
LANGUAGE js AS """
  return "Hello, " + a + "!";
  """;
SELECT greeting(name) as everyone
FROM names
```

```
+----------------+
| everyone       |
+----------------+
| Hello, Hannah! |
| Hello, Max!    |
| Hello, Jakob!  |
+----------------+
```

BigQuery UDFs Reference

Google Cloud

Pitall: User-Defined Functions **hurt Performance**

- Use native SQL functions whenever possible
- Concurrent rate limits:
  - for non-UDF queries: 50
  - for UDF-queries: 6

[BigQuery Quota Policy]

Google Cloud

Module 10

Advanced Functions and Clauses

In this module we will:

- Introduce Advanced Functions (Statistical, Analytic, User-Defined)
- **Discuss Effective Sub-query and CTE design**

Google Cloud

Using WITH Clauses (CTEs) and Subqueries

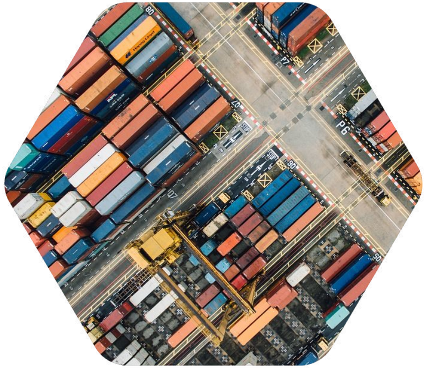
```sql
#standardSQL
#CTEs
WITH

  # 2015 filings joined with organization details
  irs_990_2015_ein AS (
  SELECT *
  FROM
    `bigquery-public-data.irs_990.irs_990_2015`
  JOIN
    `bigquery-public-data.irs_990.irs_990_ein` USING (ein)
    ),

  # duplicate EINs in organization details
  duplicates AS (
  SELECT
    ein AS ein,
    COUNT(ein) AS ein_count
  FROM
    irs_990_2015_ein
  GROUP BY
    ein
  HAVING
    ein_count > 1
    )

# return results to store in a permanent table
SELECT
  irs_990.ein AS ein,
  irs_990.name AS name,
  irs_990.noemplyeesw3cnt AS num_employees,
  irs_990.grsrcptspublicuse AS gross_receipts
  # more fields ommited for brevity
FROM irs_990_2015_ein AS irs_990
LEFT JOIN duplicates
ON
  irs_990.ein=duplicates.ein
WHERE
  # filter out duplicate records
  duplicates.ein IS NULL
```

- WITH is simply a **named subquery** (or Common Table Expression)
- Acts as a temporary table
- Breaks up complex queries
- Chain together multiple subqueries in a single WITH
- You can reference other subqueries in future subqueries

[BigQuery WITH Clause]

Google Cloud

Summary: Answer more complex questions with advanced SQL



Consider using approximation functions for really large datasets

Operate over sub-groups of rows with analytical window functions

User-defined functions add sophistication at the expense of performance

Break apart complex questions into steps with `WITH` and temporary tables

Google Cloud

Lab 9
Deriving Insights with Advanced SQL Functions

Deriving Insights with Advanced SQL Functions

In this lab, you will explore Deriving Insights from Advanced SQL Functions

```
WITH temp_table AS (
...
)

SELECT * FROM temp_table
```


Google Cloud