Journal of Electrical Engineering and Information Technologies, Vol. **7,** No. 2, pp. 77–85 (2022)
Article 198 In print: ISSN 2545–4250
Received: October 29, 2022 On line: ISSN 2545–4269
Accepted: November 15, 2022 UDC: 004.62.057.8-049.7
DOI: https://doi.org/10.51466/JEEIT2272198077t

Original scientific paper

BIGQUERY FOR BIG DATA ANALYSIS

Žaneta Trenčeva, Aleksandar Risteski, Toni Janevski, Borislav Popovski

Faculty of Electrical Engineering and Information Technologies,
"Ss. Cyril and Methodius" University in Skopje,
Rugjer Bošković bb, P.O. Box 574, 1001 Skopje, Republic of North Macedonia
zanetatrenceva1@yahoo.com

A b s t r a c t: In todays digital era, enormous amounts of data from various sources are generated daily. This data, also known as big data, is too complex to be managed using traditional data management systems. As a result, many technologies capable of handling big and complex data have emerged in the industry. One of them is Google Cloud's BigQuery. Designed to overcome the problem of traditional databases, the BigQuery platform offers storage and analysis of big data, while providing high scalability and reliability. We will be using BigQuery to gain insights into the content ratings of an OTT (Over-The-Top) TV platform.

Key words;; analysis; content; data; query; storage

BIGQUERY ЗА АНАЛИЗА НА ГОЛЕМИ ПОДАТОЦИ

А п с т р а к т: Во денешната дигитална ера огромни количества податоци од најразлични извори се генерираат на дневно ниво. Овие податоци, познати и како „големи податоци", се премногу комплексни за да бидат менаџирани користејќи традиционални системи за управување со податоци. Како резултат, во индустријата се појавија многу технологии способни да се справат со обемни и комплексни податоци. Една од нив е BigQuery на Google Cloud. Дизајнирана да го надмине проблемот на традиционалните бази на податоци, платформата BigQuery нуди складирање и анализа на големи податоци, притоа обезбедувајќи висока скалабилност и доверливост. Ние ќе го користиме BigQuery за да добиеме увид во гледаноста на содржините на една OTT (Over-The-Top) телевизиска платформа.

Клучни зборови: анализа; содржина; податоци; барање; складирање

1. INTRODUCTION

In the past, solutions for big data management were not simple or cheap. Not only did businesses need to make a huge upfront investment in hardware and software, they also had to bring experts in data analytics into their staff too. Today, the huge amount of data in any business has forced companies to look for new, innovative solutions to this problem. One of them is Google's BigQuery, a fully managed, cloud-based serverless data warehouse. Essentially, the system works by supporting analytics strategies in a huge-scale data environment.

BigQuery is a fully managed enterprise data warehouse designed to help organizations manage and analyze their data with built-in features like machine learning, geospatial analysis, and business intelligence. BigQuery's serverless architecture lets its users use SQL queries to answer their organization's biggest questions with no infrastructure management. Its scalable, distributed analysis engine can provide querying terabytes in seconds and petabytes in minutes.

Big Query interfaces include Google Cloud console interface and the BigQuery command-line tool. Developers and data scientists can use client libraries with familiar programming including Python, Java, JavaScript, and Go, as well as BigQuery's REST API and RPC API to transform and manage data.

This paper is organized in the following manner. Section 2 provides an overview of prior related work on this topic. Section 3 describes the principles of operation of BigQuery, namely the technologies and algorithams it uses to handle Big Data. The topic of section 4 are the BigQuery concepts, that is, how the data stored in BigQuery is structured, what operations can be run on it, what data

types are supported and so on. These concepts are explained by making comparisons with traditional relational databases, such as MySQL, due to the similarities between the two types of systems. Section 5 is a demonstration of a practical usage of BigQuery, using simulated data from an OTT (Over-The-Top) TV application, resulting in insights into the ratings of its content. In section 6, the usage of the PHP Client library for the BigQuery API is presented, to demonstrate how data stored in BigQuery can be accessed from a PHP web application. Section 7 concludes this paper.

2. RELATED WORK

There are many research papers regarding BigQuery for big data manipulation. In [1], a simple approach of using BigQuery for storing and analyzing data is illustrated. In the study, data samples in CSV format are taken from a publicly available pool and imported into BigQuery. Then, this data is queried from the GCP (Google Cloud Platform) console, where the results are shown as well. Reference [2] presents a method of managing and handling non-relational data in BigQuery and calculating the execution time of queries. This paper only covers the analysis time with the dataset's size using Google SDK (Software Development Kit) rather than extracting the taken dataset's necessary values. Reference [3] validates the use of big data and cloud technologies in education related analytical applications, which are also called educational intelligence applications. It presents a prototype, which is a modified version of an open-source tool called BigQuery Visualizer. The prototype is a web application that is used to make queries to a BigQuery dataset and create plots and graphs for analytical applications. Reference [4] details the functionality of edx2bigquery – an open source Python package developed by Harvard and MIT to ingest and report on hundreds of course datasets from edX (an online course provider created by Harvard and MIT), making use of BigQuery to handle multiple terabytes of learner data. The authors find that BigQuery provides ease of use in loading the multifaceted MOOC (Massive Open Online Course) datasets and near real-time interactive querying of data, including large clickstream datasets. Moreover, flexible research and reporting dashboards are provided by visualizing and aggregating data, using services associated with BigQuery. In [5] the authors present and evaluate a novel and efficient RDF (Resource Description Framework) dictionary compression algorithm, where BigQuery is used to store and query the compressed data. The proposed algorithm is faster, generates small dictionaries that can fit in memory and results in better compression rate when compared with other large scale RDF dictionary compression algorithms. Consequently, it reduces the BigQuery storage and query costs. Reference [6] offers an overview of Explainable AI in BigQuery ML, using as an example a (fictional) realtor's linear regression model that predicted a home's latest sale price based on predictor variables such as the total tax assessment from the year of the last sale, the square footage of the house, the number of bedrooms, the number of bathrooms, and whether the condition of the home is below average. After training the linear model, the feature attribution can be studied from a global and local perspective in BigQuery.

3. PRINCIPLES OF OPERATION

BigQuery can handle a sheer amount of data while looking mostly like any other SQL database (like MySQL). How can BigQuery do what MySQL cannot? We will start by looking at the problem's two parts. First, if we need to filter billions of rows of data, we need to do billions of comparisons, which require a lot of computing power. Second, we need to do the comparisons on data that is stored somewhere, and the drives that store that data have limits on how quickly it can flow out of them to the computer that is doing those comparisons. Those two problems are the fundamental issues that need to be solved, so we will look at how BigQuery tries to address each of them [7].

a) *Scaling computing capacity*

People originally tackled the computation aspect of this problem by using the MapReduce algorithm, where data is chopped into manageable pieces and then reduced to a summary of the pieces. This speeds up the entire process by parallelizing the work to lots of different computers, each working on some subset of the problem. For example, if we had a few billion rows and wanted to count them, the traditional way to do this would be to run a script on a computer that iterates through all the rows and keeps a counter of the total number of rows, which would take a long time. Using MapReduce, we could speed this up by using 1,000 computers, with each one responsible for counting one one-thousandth of the rows, and then summing up the 1,000 separate counts to get the full count (Figure 1).

In short, this is what BigQuery does under the hood. Google Cloud Platform has thousands of CPUs in a pool dedicated to handling requests from BigQuery. When we execute a query, it momentarily gives us access to that computing capacity, with each unit of computing power handling a small

piece of the data. Once all the little pieces of work are done, BigQuery joins them all back together and gives us a query result.

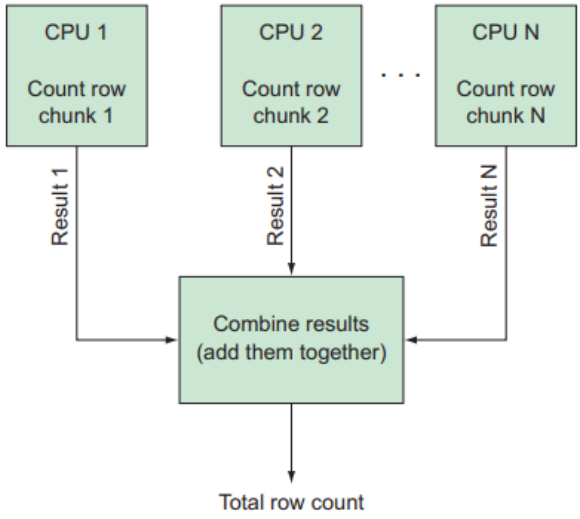


Fig. 1. Counting a few billion rows by breaking them into chunks

b) *Scaling storage throughput*

When we solved the computational capacity problem by splitting the problem up into many chunks and using lots of CPUs to crunch on each piece in parallel, we never thought about how we would make sure all of the CPUs had access to the chunks of data. If these thousands of CPUs all requested the data from a single hard drive, the drive would get overwhelmed in no time. The problem is compounded by the fact that the total amount of data you need to query is potentially enormous.

To make this more concrete, most drives, regardless of capacity, typically can sustain hundreds of megabytes per second of throughput. At that rate, pulling all the data off of one 10-terabyte (TB) drive (assuming a 500 MB/s sustained transfer rate) would take about five hours. If 1,000 CPUs all asked for their chunk of data (1,000 chunks of 10 GB each), it would take about five hours to deliver them, with a best case of about 20 seconds per 10 GB chunk. The single disk acts as a bottleneck because it has a limited data transfer rate.

To fix this, the database could be splitted across lots of different physical drives (called "sharding") (Figure 2) so that when all of the CPUs started asking for their chunks of data, lots of different drives would handle transferring them. No drive alone would be able to ship all the bytes to the CPUs, but the pool of many drives could ship all that data quickly. For example, if we were to take those same 10 TB and split them across 10,000 separate drives, 1 GB would be stored on each drive. Looking at the fleet of all the drives, the total throughput available would be around 5 TB/s. Also, each drive could ship the 1 GB it was responsible for in around two seconds. Regarding the example with 1,000 separate CPUs each reading their 10 GB chunk (one one-thousandth of the 10 TB), they would get the 10 GB in two seconds-each one would read ten 1 GB chunks, with each chunk coming from one of 10 different drives.

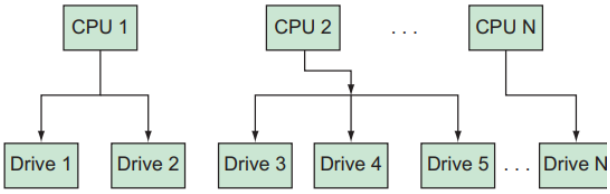


Fig. 2. Sharding data across multiple disks

Under the hood, Google is doing this, using a custom-built storage system called Colossus, which handles splitting and replicating all of the data.

4. CONCEPTS

As already mentioned, BigQuery is very SQL-like, so close comparisons can be drawn with things the reader is most probably familiar with in systems like MySQL [7].

a) *Datasets and tables*

Like a relational database has databases that contain tables, BigQuery has datasets that contain tables (Figure 3). The datasets mainly act as containers, and the tables, again like a relational database, are collections of rows. Unlike a relational database, the user does not necessarily control the details of the underlying storage systems, so although datasets act as collections of tables, one has less control over the technical aspects of those tables than they would with a system like MySQL or PostgreSQL.

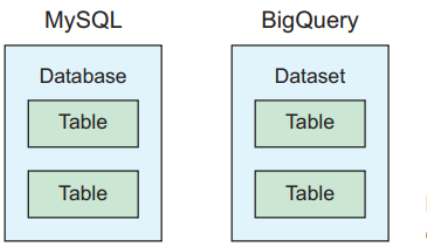


Fig. 3. A BigQuery dataset and tables compared to a MySQL database and tables

Each table contained in the dataset is defined by a set schema, so BigQuery can be thought of in a traditional grid, where each row has cells that fit the types and limits of the columns defined in the schema. It gets a little more complicated than that when a particular column allows nested or repeated values, but we will explore that in more detail later in this paper.

Unlike in a traditional relational database, BigQuery rows typically do not have a unique identifier column, primarily because BigQuery is not meant for transactional queries where a unique ID is required to address a single row. Because BigQuery is intended to be used as an analytical storage and querying system, constraints like uniqueness in even a single column are not available. Otherwise, BigQuery will accept most common SQL-style requests, like SELECT statements, UPDATE, INSERT, and DELETE statements with potentially complex WHERE clauses, as well as JOIN operations.

b) *Schemas*

As with other SQL databases, BigQuery tables have a structured schema, which in turn has the standard data types, such as INTEGER, TIMESTAMP, and STRING (sometimes known as VARCHAR). Additionally, fields can be required or nullable (like NULL or NOT NULL). Unlike with a relational database, we define and set schemas as part of an API call rather than running them as a query.

For example, we might have a table of people with fields for each person's name, age, and birth date, but instead of running a query that looks like CREATE TABLE, we would make an API call to the BigQuery service, passing along the schema as part of that message. We can represent the schema itself as a list of JSON objects, each with information about a single field. In the following example listing, the NULLABLE and REQUIRED (SQL's NOT NULL) are listed as the mode of the field.

```
{"name": "name",      "type": "STRING",    "mode": "REQUIRED"},
{"name": "age",       "type": "INTEGER",   "mode": "NULLABLE"},
{"name": "birthdate", "type": "TIMESTAMP", "mode": "NULLABLE"}
```

There is an additional mode called REPEATED, which is currently not common in most relational databases. Repeated fields do as their name implies, taking the type provided and turning it into an array equivalent. A repeated INTEGER field acts like an array of integers. BigQuery comes with special ways of decomposing these repeated fields, such as allowing us to count the number of items in a repeated field or filtering as long as a single entry of the field matches a given value.

Next, a field type called RECORD acts like a JSON object, allowing us to nest rows within rows. For example, the people table could have a RECORD type field called favorite_book, which in turn would have fields for the title and author (which would both be STRING types). Using RECORD types like this is not a common pattern in standard SQL, where it would be normalized into a separate table (a table of books, and the favorite_book field would be a foreign key). In BigQuery, this type of inlining or denormalizing is supported and can be useful, particularly if the data (in this case, the book title and author) is never needed in a different context – it is only ever looked at alongside the people who have the book as a favorite.

5. THE TV PROJECT ON GCP CONSOLE

For our practical demonstration of BigQuery, we will be using simulated data from an OTT TV platform, an interactive TV service that allows users to watch live TV, VOD (Video On Demand), record programs and more. The TV application, which is an Android application, is integrated with Firebase, and it uses the Google Analytics for Firebase SDK. Google Analytics is an application measurement solution, that provides insight on application usage and user engagement. The SDK automatically captures a number of events and user properties, but also allows users to define their own custom events to measure the things that uniquely matter to their business. Automatically collected events are triggered by basic interactions with the application, and no additional code should be written to collect them. Some automatically collected events include: "first_open" (the first time a user launches an application after installing or re-installing it), "user_engagement" (when the application is in the foreground for at least one second), "dynamic_link_app_open" (when a user re-opens the application via a dynamic link), etc.

To the contrary, custom events are defined by the developer of the application, by explicitly writing code in the desired places in the program, that will include the custom event name and (optionally) custom event parameters. In our practical example, we will be working with custom events.

When users watch content on our OTT TV platform, they generate a large amount of different events. In the application, there is code that "catches" these user interaction events that can be, for example, changing a channel, scheduling a recording of a program, interacting with a menu, pausing or rewinding a live channel, etc. All this data,

that in fact represents user behaviour, is available in Google Analytics, and since the Firebase project is linked to BigQuery, it is also stored in a dedicated project in BigQuery.

We can open our BigQuery project in the GCP console. The console provides a graphical interface used to create and manage BigQuery resources and run SQL queries.

Figure 4 is a screenshot of the "SQL workspace" section of the BigQuery page of our project in the console. It consists of an Explorer pane and a Details pane.

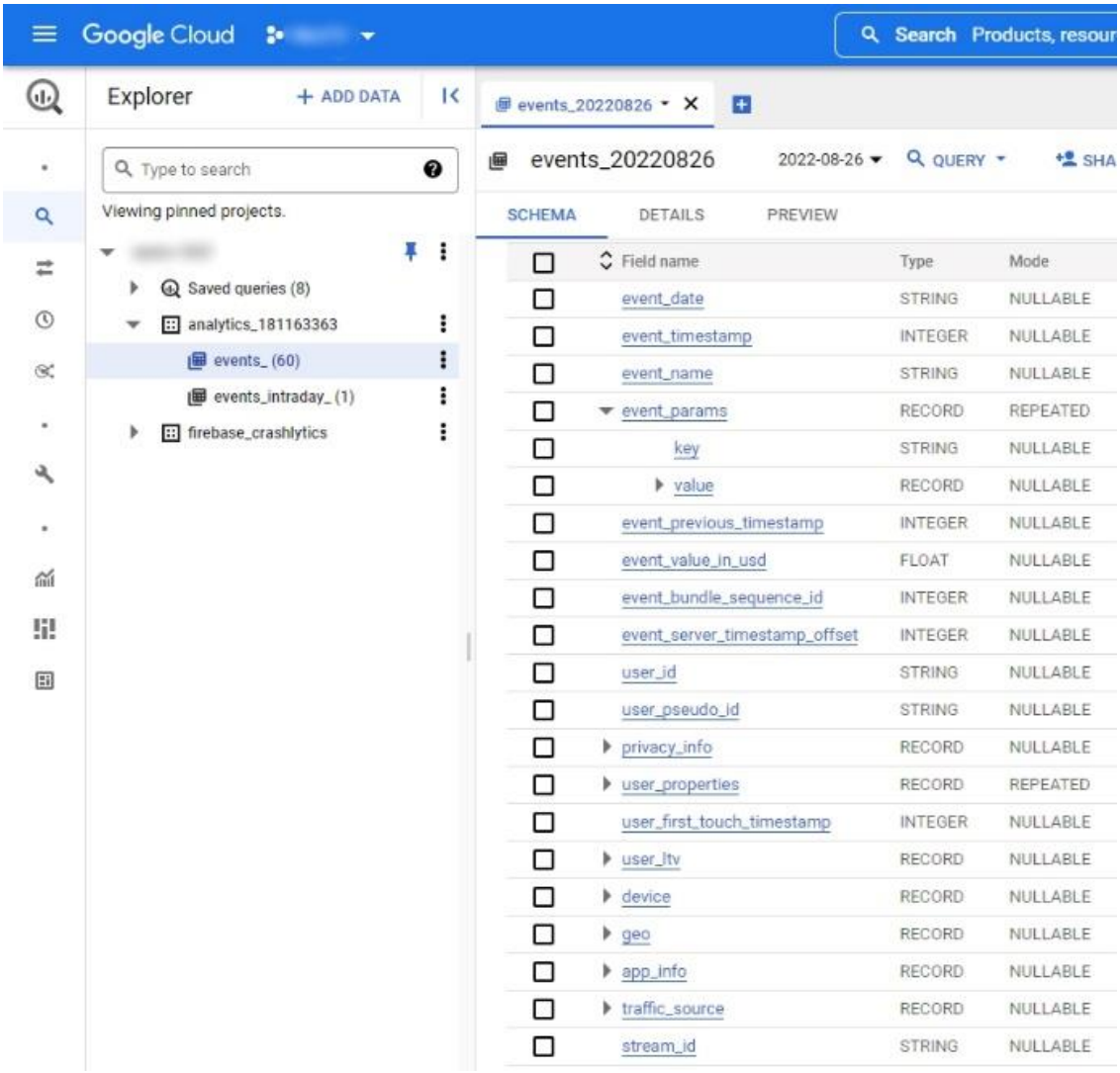


Fig. 4. The BigQuery dashboard of the TV project

The Explorer pane lists current Cloud projects and any pinned projects. Datasets can be accessed by expanding the project, and tables, views and functions can be accessed by expanding the dataset.

The Details pane shows information about the BigQuery resources. When we select a dataset, table, view, or other resource in the Explorer pane, a new tab is displayed. On these tabs, we can view information about the resource, create tables and views, modify table schemas, execute SQL queries, export data, and perform other actions.

For each Firebase project that is linked to BigQuery, a single dataset named "analytics_<property_id>" is added to the BigQuery project. Property ID refers to the Analytics Property ID, which can be found in App Analytics Settings in the Firebase project.

Regarding dataset structure, within each dataset, a table named "events_YYYYMMDD" is created each day whether the Daily export or Streaming export option is enabled (during the Firebase integration process). Moreover, if the Streaming export option is enabled, an additional table, "events_intraday_YYYYMMDD", is created. This table is populated continuously as events are recorded throughout the day. It is deleted at the end of each day once "events_YYYYMMDD" is complete.

That being said, we can now explore our project. The project contains two datasets, "analytics_181163363" and "firebase_crashlytics", as well as several saved queries.

We are interested in the "analytics_181163363" dataset. It contains 60 tables named "events_YYYYMMDD", which store events that happened on the specific date in the past 60 days. There is also the "events_intraday_YYYYMMDD" table, which stores events happening on the current day, as we previously explained. Figure 4 also shows a part of these tables' schema, that is, the fields (table columns), their types and their modes. For example, the field "event_name" is of type STRING and it is NULLABLE. The field "event_params", on the other hand, is of type RECORD, and it is in REPEATED mode. That means that it acts as an array of struct types, that is, there can be many entries within this field, and each of them will have both "key" and "value".

By examining the table's fields, we can conclude that these tables store various information, such as information about the triggering event, the user, the user device, the application, geo location etc. The schema of these tables, that is, the fields, is defined by Google Analytics, and not by the user. The user can, however, specify event parameters for their custom events.

Next to the Schema tab (that we have been examining so far) in the Explorer pane, is the Details tab, which gives some basic table information, such as its size, number of rows, expiration date etc. It is shown on Figure 5. The last tab is the Preview tab, where actual table entries are shown, sorted in descending order. Two entries with only the first several fields can be seen on Figure 6.

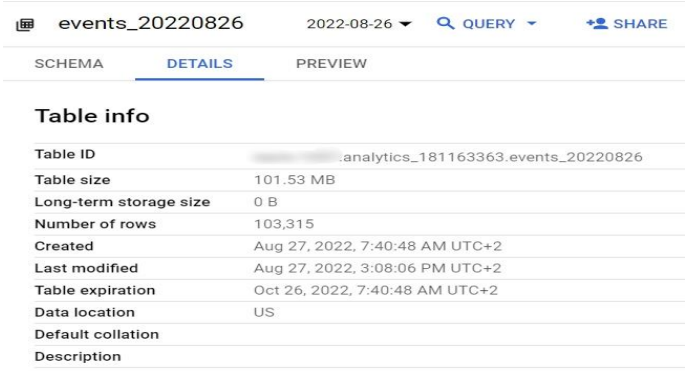


Fig. 5. Details tab of the Explorer pane

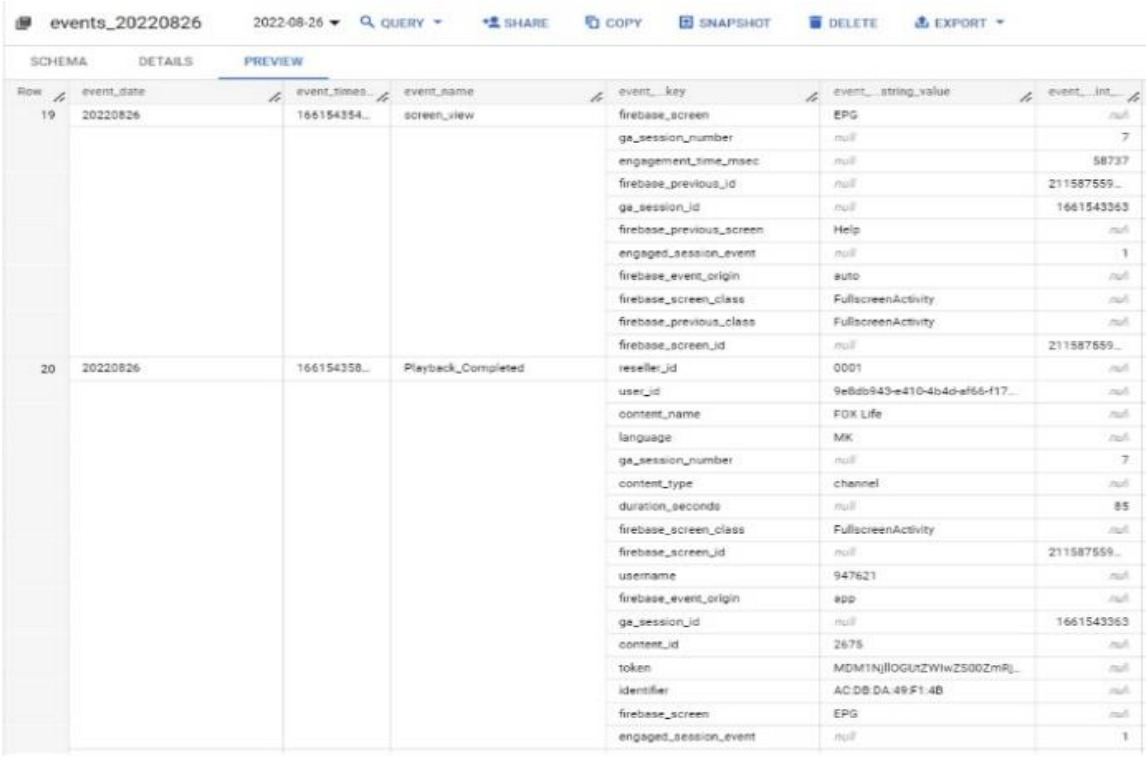


Fig. 6. Preview tab of the Explorer pane

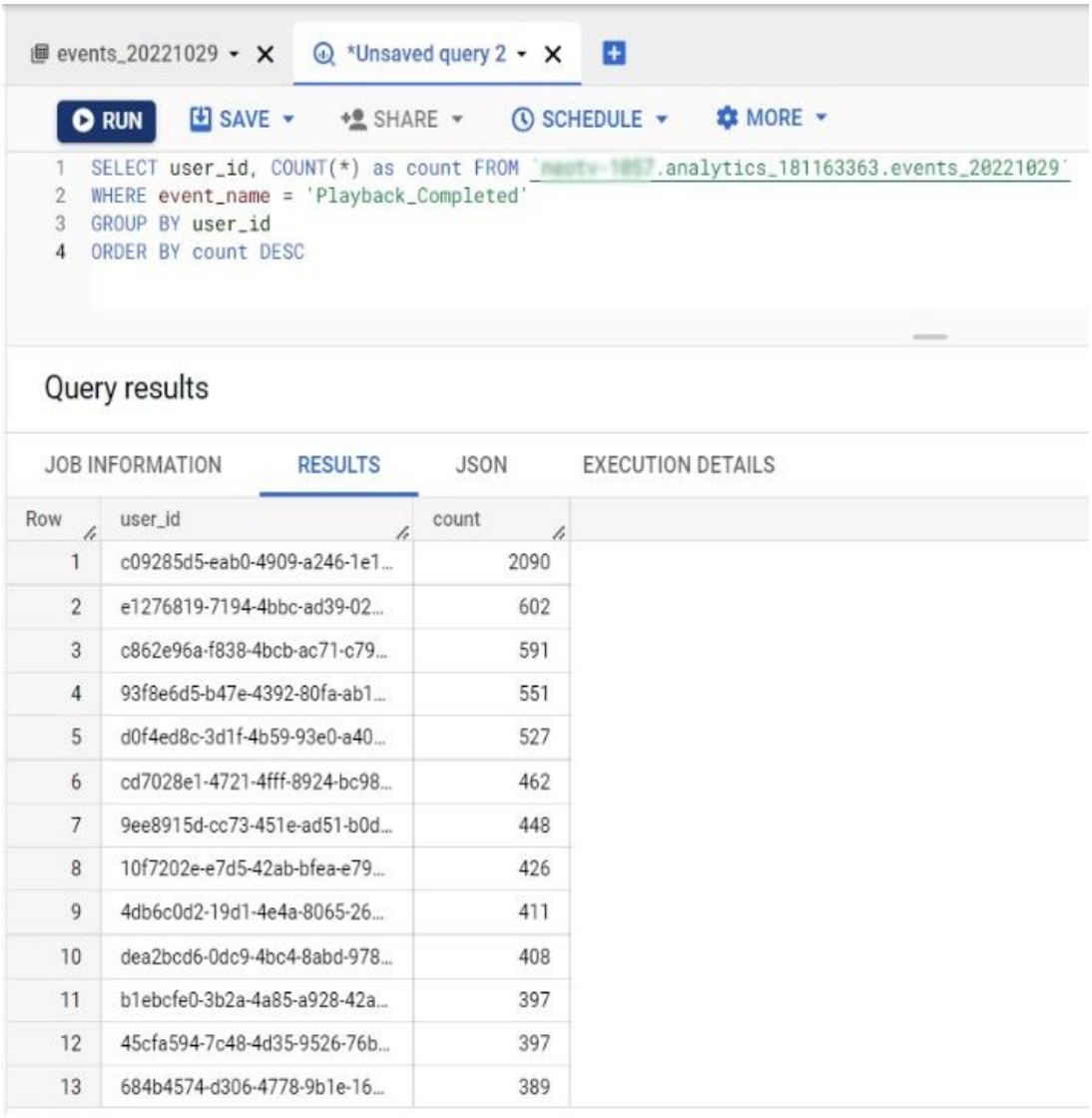
We will now demonstrate query execution in the console with two examples. In both of them we are working with custom events whose “event_name” parameter is “Playback_Completed”, as those are the events that hold data about user sessions (and therefore, content ratings).

The first query and part of its results are shown on Figure 7. The query uses the events table from 29.10.2022, and filters records whose “event_name” field is “Playback_Completed”. It then groups these records by user ID, counts them, and displays the results in descending order of the number of records.

Figure 8 shows another example of a SQL query in the console and part of its results upon execution.

In the SQL query, it can be noted that an UNNEST operator is being used. The UNNEST operator is used to convert an array into set of rows, also known as “flattening”. It takes an array and returns a table with a single row for each element in it. Basically, we are unnesting the “event_params” field (which is an array, as its mode is REPEATED) by key (“content_name”, “duration_seconds”, “content_type” and “username”).

What the query does is, it fetches results in the form of content ratings cumulatively, namely how much time a certain content was watched, and how many sessions of it occurred. It does this the following way: it uses the “Playback_Completed” events (that is, user sessions), filtered by a starting date and an ending date, the type of content watched (in this case, live channel, but can also be timeshift, VOD and radio) and grouped by the content name. Then it sums the “duration_seconds” parameter of each session to calculate the overall ratings in seconds (hours) of the given content, and counts the “content_name” parameter (or any other from the “event_params” field) to determine the number of times the given content was watched. Finally, the data is ordered in descending order by the overall time a content was watched.



Row	user_id	count
1	c09285d5-eab0-4909-a246-1e1...	2090
2	e1276819-7194-4bbc-ad39-02...	602
3	c862e96a-f838-4bcb-ac71-c79...	591
4	93f8e6d5-b47e-4392-80fa-ab1...	551
5	d0f4ed8c-3d1f-4b59-93e0-a40...	527
6	cd7028e1-4721-4fff-8924-bc98...	462
7	9ee8915d-cc73-451e-ad51-b0d...	448
8	10f7202e-e7d5-42ab-bfea-e79...	426
9	4db6c0d2-19d1-4e4a-8065-26...	411
10	dea2bcd6-0dc9-4bc4-8abd-978...	408
11	b1ebcfe0-3b2a-4a85-a928-42a...	397
12	45cfa594-7c48-4d35-9526-76b...	397
13	684b4574-d306-4778-9b1e-16...	389

Fig. 7. Query execution and results: Number of sessions by user ID

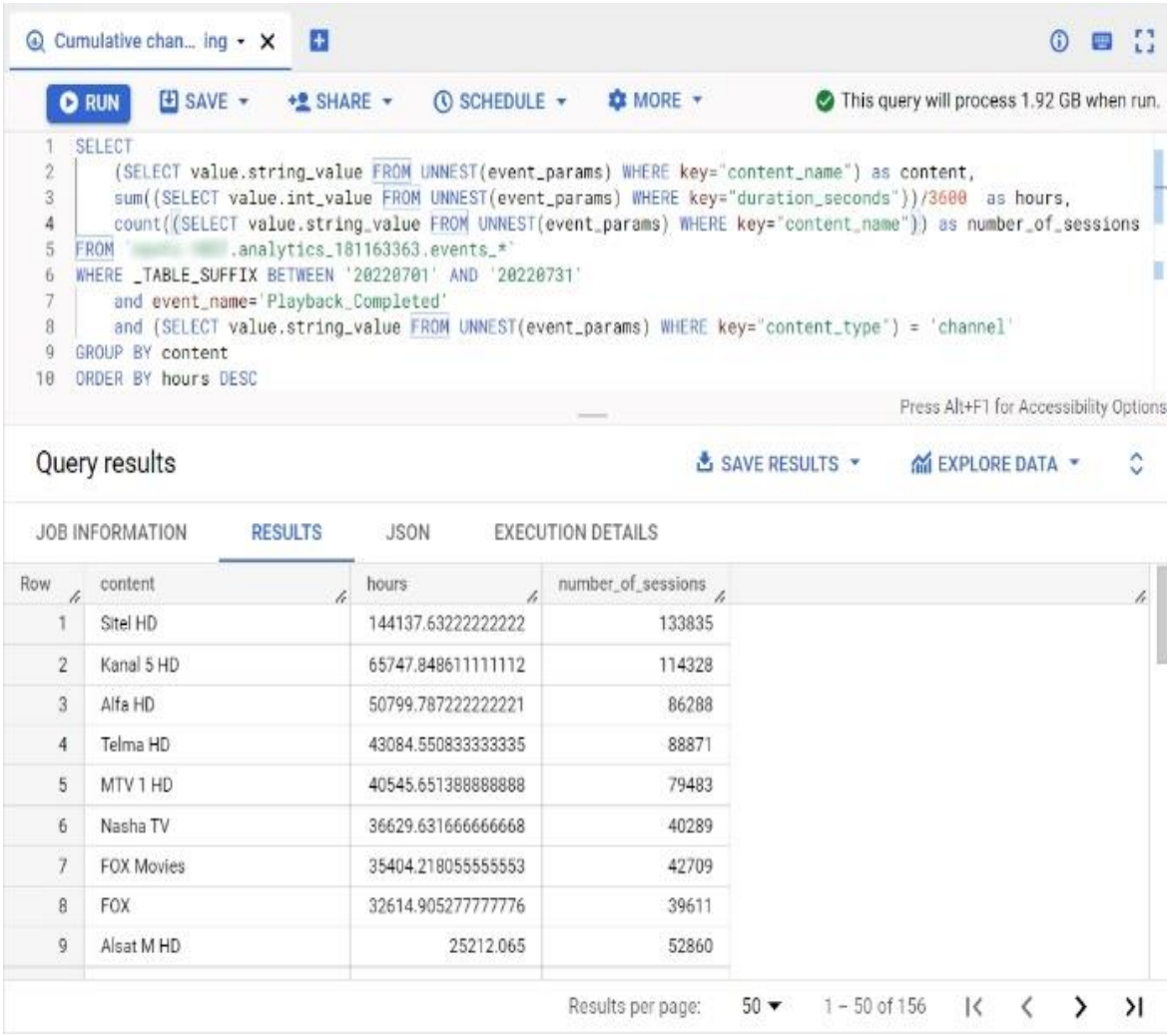


Fig. 8. Query execution and results: Watch time and number of sessions of live channels

As we can see, "Sitel" is the most watched live TV channel, with approximately 144 137 hours of watch time and 133 835 user sessions. We can visualize these results using Google Data Studio, an online tool for converting data into customizable informative reports and dashboards. We can do that by clicking the export data button and then choosing Data Studio. Figure 9 shows (a part of) the query results presented in a bar chart.

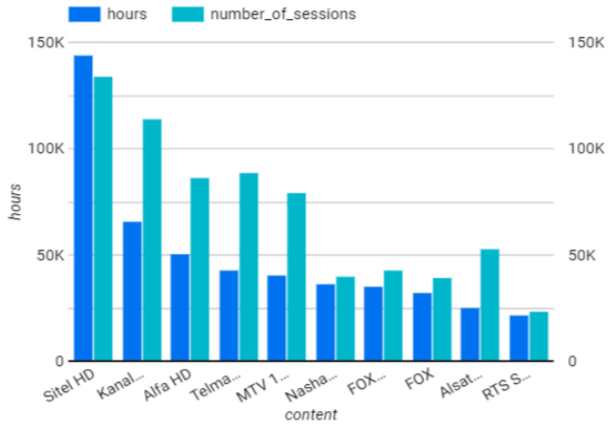


Fig. 9. Bar chart of query results in Data Studio

6. THE PHP CLIENT LIBRARY FOR THE BIGQUERY API

Google Cloud APIs are programmatic interfaces to Google Cloud Platform services, that allow users to easily add different functionalities to their applications.

Client libraries make it easier to access Google Cloud APIs from a supported language. While Google Cloud APIs can be used directly by making raw requests to the server, client libraries provide simplifications that significantly reduce the amount of code that needs to be written.

We will now take a look at how to use the PHP Client library for the BigQuery API, that is, how to access data stored in a BigQuery project from a PHP web application.

Assuming we have a PHP web application, we first need to install the client library in it. In PHP, this is done by running "composer require google/cloud-bigquery". In order to do that, we must first set up authentication. One way to set up authentication is to create a service account in the TV

project in the cloud console and then generate a service account key and download it locally. Finally, we need to make a connection to the TV project, using the generated key. The code for this purpose is the following, where the JSON file is the downloaded key:

```
<?php
require 'vendor/autoload.php';
use Google\Cloud\BigQuery\BigQueryClient;
$projectId = '<project-name>';
$path = '<project-name>-d0425aeab4c1.json';
$bigQuery = new BigQueryClient([
'projectId' => $projectId,
'keyFilePath' => $path,
]);
?>
```

The "bigQuery" variable holds the connection to the project in BigQuery and will be used to execute the SQL queries.

If we have a SQL query like the one we saw earlier in this paper, and assign it to a STRING variable called "query", we can then execute the query programmatically, using the following code:

```
$queryJobConfig = $bigQuery->query($query);
$queryResults = $bigQuery -> runQuery($query JobConfig);
```

If the query has run successfully, we have the results in the "queryResults" variable. Finally, we are extracting the "content", "hours" and "no_sessions" fields of the resulting rows and simply echoing them to the web page. The code for this purpose is the following:

```
if ($queryResults->isComplete()) {
    $rows = $queryResults->rows();
    $results = array();
    foreach ($rows as $row) {
     echo $row['content'] . ' ' . $row['hours'] . ' ' . $row['no_sessions']);
    } else {
    throw new Exception('The query failed to complete');
}
```

7. CONCLUSION

Google BigQuery is a service designed to provide its customers with insight into their businesses quickly and cost-effectively. With a company's data system located on the cloud, comes the freedom and flexibility to modernize its entire business structure. Some of the most significant advantages offered by BigQuery are:

– Accelerated time to value: Users can gain insight into their businesses as soon as they start using the service (no prior planning and implementation costs).

– Simplicity and scalability: All analytical requirements can be performed through a simple and effective interface, without additional management infrastructure. The system can scale depending on demand for performance, size and pricing.

– Speed: With BigQuery, data is processed at a tremendous speed thanks to the technologies it uses.

– Security: All projects are encrypted and protected with IAM (Identity and Access aManagement) support.

– Reliability: Google Cloud and BigQuery enable access to always-on servers, and geographic replication across a huge selection of Google data centers around the world.

Throughout this paper, we saw the benefits of BigQuery in action. The example solution we presented for storage and analysis of OTT TV platform data can prove very useful for business analysts of the platform, as it provides information on which specific content has the most watch time and which has the least.

REFERENCES

[1] Ali, H., Hosain, S., Hossain, A. (2021): Big Data analysis using BigQuery on cloud computing platform, *Australian Journal of Engineering and Innovative Technology*, **3** (1), pp. 1–9.

[2] Kotecha, B., Joshiyara, H. (2018): Handling non-relational databases on Big Query with scheduling approach and performance analysis", *2018 Fourth International Conference on Computing Communication Control and Automation (ICCUBEA)*, pp. 118–127.

[3] Khan, S., Alam, M. (2018): Analyzing Big 'Education' Data using BigQuery and R, *8th DBT-BIF National Workshop on Translational Bioinformatics: Bench-to-Bedside*, Department of Computer Science, April 9–10;

[4] Lopez, G., Seaton, D., Ang, A., Tingley, D., Chuang, I. (2017): Google BigQuery for education: Framework for parsing and analyzing edX MOOC data, *Proceedings of the Fourth (2017) ACM Conference on Learning @ Scale*, pp. 181–184.

[5] Dawelbeit, O., Mccrindle, R. (2016): Efficient dictionary compression for processing RDF Big Data using Google BigQuery, *2016 IEEE Global Communications Conference.* 10.1109/GLOCOM.2016.7841775.

[6] Lichtendahl, K. C., Boatright, B. (2022): Google Cloud Platform: BigQuery Explainable AI, Darden Case No. UVA-QA-0943.

[7] Geewax, J. J. (2018): *Google Cloud Platform in Action*, Manning Publications, USA.