Amazon Redshift to BigQuery migration guide

About this document

The information and recommendations in this document were gathered through our work with a variety of clients and environments in the field. We thank our customers and partners for sharing their experiences and insights.

Takeaways	• Strategies for migration • Query optimization and data modeling best practices • Troubleshooting tips • User adoption guidance

Highlights	
Purpose	To provide high-level guidance for organizations that are migrating Redshift to BigQuery.
Intended audience	Enterprise architects, DBAs, application developers, and IT security.
Key assumptions	That the audience is familiar with Redshift and is looking for guidance on transitioning to BigQuery.

Objectives of this guide

1. Provide high-level guidance to organizations migrating from Redshift to BigQuery.
2. Show ways organizations can rethink their existing data pipelines to get the most out of BigQuery.
3. Compare the architectures of BigQuery and Redshift to help organizations map features and capabilities during migration. Our goal is to show you new capabilities available to your organization through BigQuery, not to map features one-to-one with Redshift.

Nonobjectives of this guide

1. Provide detailed instructions for all activities required to migrate from Redshift to BigQuery.
2. Present solutions for every use case of Redshift in BigQuery.
3. Replace our BigQuery documentation or act as a substitute for training on big data. We strongly recommend that you take our Big Data and Machine Learning classes and read our extensive documentation prior to, or in conjunction with, this guide.

Introduction

This document and its companion document Amazon Redshift to BigQuery SQL translation reference are part of the enterprise data warehouse (EDW) migration initiative and highlight technical differences and provide guidance on migrating from the Amazon Redshift data warehouse to BigQuery, Google's fully managed, petabyte-scale data warehouse. This document specifically targets migrations of EDW and online analytical processing (OLAP) use cases. If you need online transaction processing (OLTP) behavior like single-row updates or inserts, consider a database designed to support OLTP use cases such as Cloud SQL.

Premigration tasks

To help ensure a successful data warehouse migration, start planning your migration strategy early in your project timeline. This lets you evaluate the Google Cloud features that suit your needs.

BigQuery capacity planning

Under the hood, analytics throughput in BigQuery is measured in slots. A BigQuery slot is Google's proprietary unit of computational capacity required to execute SQL queries. BigQuery continuously *calculates* how many slots are required by queries as they execute, but it *allocates* slots to queries based on a fair scheduler.

You can choose between the following pricing models when capacity planning for BigQuery slots:

- **On-demand pricing:** Under on-demand pricing, BigQuery charges for the number of bytes processed (data size), so you pay only for the queries that you run. For more information about how BigQuery determines data size, see Data size calculation. Because slots determine the underlying computational capacity, you can pay for BigQuery usage depending on the number of slots you'll need (instead of bytes processed). As a side note, by default all Google Cloud projects are limited to a

maximum of 2000 slots. Note that BigQuery might burst beyond this limit to accelerate your queries, but bursting is not guaranteed.

- **Flat-rate pricing:** Instead of paying for bytes processed by queries that you run, you purchase BigQuery slot reservations (a minimum of 100) in monthly or yearly commitments. Flat-rate is the recommended pricing model for EDW workloads, which commonly see many concurrent reporting and ELT queries that have predictable consumption. Additionally, if your workload varies in bursts and you are looking for something short-term, flex slots offer a minimum commitment duration of 60 seconds, which you can cancel anytime.

To help with slot estimation, we recommend setting up BigQuery monitoring using Cloud Monitoring and analyzing your audit logs using BigQuery. Many customers use Google Data Studio (here's an open source example of a Data Studio dashboard), Looker, or Tableau as front ends to visualize BigQuery's audit log data, specifically for slot usage across queries and projects. You could also leverage BigQuery's system tables data for monitoring slot utilization across jobs and reservations (here's an open source example of a Data Studio dashboard). Regularly monitoring and analyzing your slot utilization helps you estimate how many total slots your organization needs as you grow on Google Cloud.

For example, suppose you initially reserve 4,000 BigQuery slots to run 100 medium-complexity queries simultaneously. If you notice high wait times in your queries' execution plans, and your dashboards show high slot utilization, this could indicate that you need additional BigQuery slots to help support your workloads. If you want to purchase slots yourself through flex, monthly, or yearly commitments, you can get started with BigQuery Reservations using the Google Cloud Console or the bq command-line tool. For more details on workload management, query execution, and BigQuery architecture, see the in-depth overview.

For any questions regarding your current plan and option, contact your sales representative to take advantage of BigQuery Reservations.

Security in Google Cloud

The following sections describe common Redshift security controls and how you can help ensure that your data warehouse stays protected in a Google Cloud environment.

Identity and access management (IAM)

Setting up access controls in Redshift involves writing Redshift API permissions policies and attaching them to IAM identities. The Redshift API permissions provide cluster-level access, but do not provide access levels more granular than the cluster. More granular access to resources like tables or views is controlled by user accounts in the Redshift database.

BigQuery uses IAM to manage access to resources at a more granular level. The types of resources available in BigQuery are organizations, projects, datasets, tables, columns, and views. In the IAM policy hierarchy, datasets are child resources of projects. A table inherits permissions from the dataset that contains it.

To grant access to a resource, assign one or more IAM roles to a user, group, or service account. Organization and project roles affect the ability to run jobs or manage the project, whereas dataset roles affect the ability to access or modify the data inside a project.

IAM provides these types of roles:

- Predefined roles are meant to support common use cases and access control patterns.
- Primitive roles include the Owner, Editor, and Viewer roles. Predefined roles provide granular access for a specific service and are managed by Google Cloud.
- Custom roles provide granular access according to a user-specified list of permissions.

When you assign both predefined and primitive roles to a user, the permissions granted are a union of the permissions of each individual role.

Within IAM, BigQuery provides table-level access control (in beta as of August 2020). Table-level permissions determine the users, groups, and service accounts that can access a table or view. You can give a user access to specific tables or views without giving the user access to the complete dataset. For more granular access, you can also look into the column-level access control (in beta as of August 2020), which provides fine-grained access to sensitive columns using *policy tags*, or type-based classification of data.

Full-disk encryption
In addition to identity and access management, data encryption adds an extra layer of defense for protecting data. In the case of data exposure, encrypted data will not be readable.

On Redshift, encryption for both data at rest and data in transit is not enabled by default. Encryption for data at rest must be explicitly enabled when a cluster is launched or by modifying an existing cluster to use AWS Key Management Service encryption. Encryption for data in transit must also be explicitly enabled.

BigQuery encrypts all data at rest and in transit by default regardless of the source or any other condition, and this cannot be turned off. BigQuery also supports customer-managed encryption keys (CMEK) for users who want to control and manage key encryption keys in Cloud Key Management Service. You can read data-at-rest encryption and encryption-in-transit whitepapers for more details on encryption at Google Cloud.

For data in transit on Google Cloud, data is encrypted and authenticated when it moves outside of the physical boundaries controlled by Google or on behalf of Google. Inside these boundaries, data in transit is generally authenticated but not necessarily encrypted.

Data loss prevention

Compliance requirements might limit what data can be stored on Google Cloud. Cloud Data Loss Prevention (Cloud DLP) can be used to scan your BigQuery tables to detect and classify sensitive data. If sensitive data is detected, Cloud DLP deidentification transformations can mask, delete, or otherwise obscure that data.

Migration to Google Cloud: The basics

Google Cloud offers a proven, integrated, end-to-end big data solution based on years of innovation at Google that lets you capture, process, store, and analyze your data within a single platform. Using BigQuery as your data warehouse enables you to focus on finding data insights and leaves management tasks like sizing a cluster of nodes to Google.

For more details, see the EDW to BigQuery migration guide.

Migration tools

The BigQuery Data Transfer Service provides an automated tool to migrate both schema and data from Redshift to BigQuery directly. The following table lists additional tools to assist in migrating from Redshift to BigQuery:

Tools	Purpose
BigQuery Data Transfer Service	Perform an automated batch transfer of your Redshift data to BigQuery by using this fully managed service.
Storage Transfer Service	Quickly import Amazon S3 data into Cloud Storage and set up a repeating schedule for transferring data by using this fully managed service.
gsutil	Copy Amazon S3 files into Cloud Storage by using this command-line tool.
bq	Interact with BigQuery by using this command-line tool. Common interactions include creating BigQuery table schemas, loading Cloud Storage data into tables, and executing queries.

Cloud Storage client libraries	Copy Amazon S3 files into Cloud Storage by using your custom tool, built on top of the Cloud Storage client library.
BigQuery client libraries	Interact with BigQuery by using your custom tool, built on top of the BigQuery client library.
BigQuery query scheduler	Schedule recurring SQL queries by using this built-in BigQuery feature.
Cloud Composer	Orchestrate transformations and BigQuery load jobs by using this fully managed Apache Airflow environment.
Apache Sqoop	Submit Hadoop jobs by using Sqoop and Redshift's JDBC driver to extract data from Redshift into either HDFS or Cloud Storage. Sqoop runs in a Dataproc environment.

For more information on using the BigQuery Data Transfer Service, see Migrating data from Amazon Redshift.

Migration using pipelines

Your data can take different paths to BigQuery since there are several tools and services available to assist when migrating Redshift to BigQuery. While the list below is not exhaustive, it does provide a sense of the different data pipeline patterns available when moving your data.

Extract and load (EL)

- **Fully automated**: The BigQuery Data Transfer Service can automatically copy your tables' schemas and data from your Redshift cluster to BigQuery.
- **More control:** If you want more control over your data pipeline steps, you can create a pipeline using either of the following options:

Pipeline using Redshift file extracts

1. Export Redshift to Amazon S3.

2. Copy data from Amazon S3 to Cloud Storage by using any of the following:
   - Storage Transfer Service (recommended)
   - gsutil (example)
   - Cloud Storage client libraries

3. Load Cloud Storage data into BigQuery by using any of the following:
   - BigQuery Data Transfer Service
   - bq command-line tool
   - BigQuery client libraries

Pipeline using a Redshift JDBC connection

Use any of the following Google Cloud products to export Redshift data by using Redshift's JDBC driver:

- Dataflow
  - Google-provided template: JDBC to BigQuery
- Cloud Data Fusion
  - Using JDBC drivers
- Dataproc
  - Connect to Redshift through JDBC using Apache Spark
    - Apache Spark examples
  - Writing to BigQuery with Apache Spark
  - Hadoop BigQuery Connector
  - Use Sqoop and Redshift's JDBC driver to extract data from Redshift into Cloud Storage:
    - Sqoop on Dataproc example

Extract, transform, and load (ETL)

If you want to transform some data before loading into BigQuery, follow the same pipeline recommendations that are described in the Extract and Load (EL) section, adding an extra step to transform your data before loading into BigQuery.

Pipeline using Redshift file extracts

1. Follow steps 1 and 2 in the Extract and load (EL) section to export Redshift data into Cloud Storage.

2. Transform and then load your data into BigQuery by using any of the following:
   a. Dataproc
      - Reading from Cloud Storage with Apache Spark
      - Writing to BigQuery with Apache Spark
      - Hadoop Cloud Storage Connector
      - Hadoop BigQuery Connector
   b. Dataflow
      - Reading from Cloud Storage
      - Writing to BigQuery
      - Google-provided template: Cloud Storage Text to BigQuery
   c. Cloud Data Fusion
      - Reading from Cloud Storage
      - Writing to BigQuery
   d. Dataprep by Trifacta
      - Reading from Cloud Storage
      - Writing to BigQuery

Pipeline using Redshift JDBC connection

Use any of the products described in the Extract and load (EL) section, but modify your pipeline to introduce one or more steps to transform your data before writing to BigQuery.

- Dataflow
  - Clone the JDBC to BigQuery template code and modify the template to add Apache Beam transforms.
- Cloud Data Fusion
  - Transform your data using any of the CDAP plugins.
- Dataproc
  - Transform your data using either Spark SQL or your own custom code in any of the supported Spark languages (Scala, Java, Python, or R).

Extract, load, and transform (ELT)

Perform transformations on your data using BigQuery itself. Accomplish this by using any of the Extract and load (EL) options to load your data into a staging table. You then transform the data in this staging table using SQL queries that write their output into your final production table.

Change data capture (CDC)

Change data capture is one of several software design patterns used to track data changes. It's often used in data warehousing because the data warehouse is used to collate and track data and its changes from various source systems over time.

For more details about manually designing your own CDC pipeline with BigQuery, see the EDW to BigQuery migration guide.

Partner tools for data migration

There are several vendors in the extract, transform, and load (ETL) space. Refer to Google Cloud's BigQuery partner website for a list of key partners and their provided solutions.

Migration to Google Cloud: An in-depth view

BigQuery is a powerful data warehouse suitable for large-scale analytics used by all types of organizations, from startups to Fortune 500 companies. Understanding some of the key features that BigQuery offers is critical to improving performance and reducing cost.

Architecture comparison

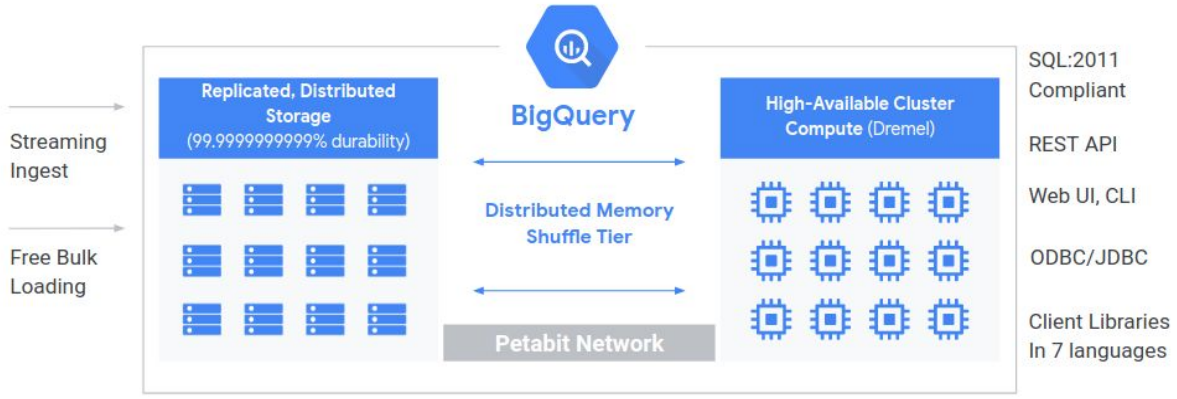
Both BigQuery and Redshift are based on a massively parallel processing (MPP) architecture. Queries are distributed across multiple servers to accelerate their execution. With regard to system architecture, Redshift and BigQuery primarily differ in how data is stored and how queries are executed. In BigQuery, the underlying hardware and configurations are abstracted away; its storage and compute scale without any intervention from the user.

Compute, memory, and storage

In Redshift, CPU, memory, and disk storage are tied together through compute nodes, as illustrated in the following diagram from the Redshift documentation. Cluster performance and storage capacity are determined by the type and the quantity of compute nodes, both of which must be configured. To change compute or storage, you need to resize your cluster through a process (over a couple of hours, or up to two days or longer) that creates a brand new cluster and copies the data over. More recently, Redshift has started to offer RA3 nodes

with managed storage that help separate compute and storage. The largest node in the RA3 category caps at 64 TB of managed storage per node.

From the start, BigQuery does not tie together compute, memory, and storage but rather treats them all separately. The following diagram illustrates BigQuery's architecture.



BigQuery compute is defined by slots, a unit of computational capacity required to execute queries. Google manages the entire infrastructure that a slot encapsulates, eliminating all but the task of choosing the proper slot quantity for your BigQuery workloads. Refer to the BigQuery capacity planning section for help deciding how many slots you'll purchase for your data warehouse. BigQuery memory is provided by a remote distributed service, connected to compute slots by Google's petabit network, all managed by Google.

BigQuery and Redshift both use columnar storage, but BigQuery uses variations and advancements on columnar storage. While columns are being encoded, various statistics about the data are persisted and later are used during query execution to compile optimal plans and to choose the most efficient runtime algorithm. BigQuery stores your data in Google's distributed file system, where it's automatically compressed, encrypted, replicated, and distributed. This is all accomplished without affecting the computing power available for your queries. Separating storage from compute allows you to scale to dozens of petabytes in storage seamlessly, without requiring additional expensive compute resources. A number of other benefits of separating compute and storage exist as well.

Scaling up or down

When storage or compute become constrained, Redshift clusters must be resized by modifying the quantity or types of nodes in the cluster.

When you resize a Redshift cluster, there are two approaches:

- **Classic resize:** Redshift creates a cluster to which data is copied, a process that can take a couple hours or as much as two days or longer for large amounts of data.
- **Elastic resize:** If you change only the number of nodes, then queries are temporarily paused and connections are held open if possible. During the resize operation, the cluster is read-only. Elastic resize typically takes 10 to 15 minutes but might not be available for all configurations.

Because BigQuery is a platform as a service (PaaS), you only have to worry about the number of BigQuery slots that you want to reserve for your organization. You reserve BigQuery slots in reservations and then assign projects to these reservations. Refer to the capacity planning section for guidance on setting up these reservations, especially the flex slots, which take seconds to deploy, start from a minimum commitment duration of 60 seconds, and can be canceled anytime.

Query execution

BigQuery's execution engine is similar to Redshift's in that they both orchestrate your query by breaking it up into steps (a query plan), executing the steps (concurrently where possible), and then reassembling the results. Redshift generates a static query plan, but BigQuery does not because it dynamically optimizes query plans as your query executes. BigQuery shuffles data using its remote memory service, whereas Redshift shuffles data using local compute node memory. For more information about BigQuery's storage of intermediate data from various stages of your query plan, see In-memory query execution in Google BigQuery.

Workload management in BigQuery

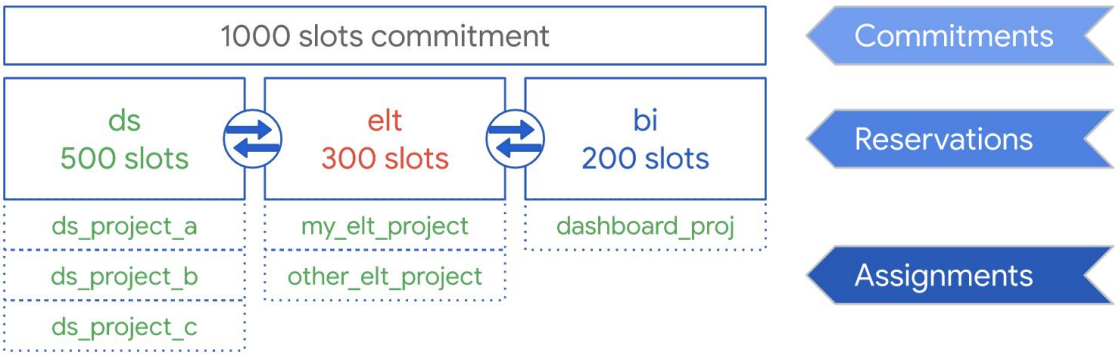
BigQuery offers the following controls for workload management (WLM):

- Interactive queries, which are executed as soon as possible (this is the default setting).
- Batch queries, which are queued on your behalf, then start as soon as idle resources are available in the BigQuery shared resource pool.
- Slot reservations through flat-rate pricing. Instead of paying for queries on demand, you can purchase BigQuery slot commitments (starting at a minimum of 100) in either flex, monthly, or yearly commitments. You can then dynamically create/manage buckets of slots called reservations and assign projects, folders, or organizations to these reservations. By default, queries running in a reservation automatically use idle slots from other reservations.

  For example, you might have purchased a total commitment capacity of 1,000 slots to share across three workload types: data science (`ds`); extract, load, transform (`elt`); and business intelligence (`bi`) (as shown in the following image).

- You can create a `ds` reservation with 500 slots, and assign all relevant Google Cloud projects to the DS reservation.
- You can create an `elt` reservation with 300 slots, and assign projects you use for ELT workloads to the `elt` reservation.
- You can create a `bi` reservation with 200 slots, and assign projects connected to your BI tools to the `bi` reservation.

Instead of partitioning your reservations across workloads, you might also choose to create reservations for individual teams or departments depending on your use case.



BigQuery's fair scheduler dynamically assigns slots to queries on an as-needed basis while maintaining fairness among multiple users who are all querying at once. For this reason, be careful when you increase the concurrent query limit because the fair scheduler evenly distributes slots across all queries up to that limit. A high concurrency limit with a low overall slot commitment leads to slower performance across all queries.

In an SQL spooling process of writing query results to a file, BigQuery creates a temporary table for approximately 24 hours to store the results of every query that runs. These results can be saved to a permanent table or downloaded to a local file.

For more information about how a temporary result set is created during the query process, see Writing query results.

Workload management in Redshift

Redshift offers two types of workload management (WLM):

- Automatic: With automatic WLM, Amazon Redshift manages query concurrency and memory allocation. Up to eight queues are created with the service class identifiers 100–107. For more information, see Query priority. Automatic WLM determines the amount of resources that queries need, and adjusts the concurrency based on the workload.

- Manual: In contrast, manual WLM requires you to specify values for query concurrency and memory allocation. The default for manual WLM is concurrency of five queries, and memory is divided equally between all five.

When concurrency scaling is enabled, Redshift automatically adds additional cluster capacity when you need it to process an increase in concurrent read queries. Concurrency scaling has certain regional and query considerations. For more information, see Concurrency scaling candidates.

Dataset and table configurations

BigQuery offers a number of ways to configure your data and tables such as partitioning, clustering, and data locality. These configurations can help maintain large tables and reduce the overall data load and response time for your queries, thereby increasing the operational efficiency of your data workloads.

Partitioning

A partitioned table is a table that is divided into segments, called partitions, that make it easier to manage and query your data. Users typically split large tables into many smaller partitions, where each partition contains a day's worth of data. Partition management is a key determinant of BigQuery's performance and cost when querying over a specific date range because it helps BigQuery scan less data per query. For examples of partitioned table queries, see the EDW to BigQuery migration guide.

There are three types of table partitioning in BigQuery:

- Tables partitioned by ingestion time: Tables are partitioned based on the data's ingestion time.
- Tables partitioned by column: Tables are partitioned based on a TIMESTAMP or DATE column.
- Tables partitioned by integer range: Tables are partitioned based on an integer column.

A column-based time-partitioned table obviates the need to maintain partition awareness independent from the existing data filtering on the bound column. Data written to a column-based time-partitioned table is automatically delivered to the appropriate partition based on the value of the data. Similarly, queries that express filters on the partitioning column can reduce the overall data scanned, which can yield improved performance and reduced query cost for on-demand queries.

BigQuery column-based partitioning is similar to Redshift's column-based partitioning, with a slightly different motivation. Redshift uses column-based key distribution to try to keep related data stored together within the same compute node, ultimately minimizing data

shuffling that occurs during joins and aggregations. BigQuery separates storage from compute, so it leverages column-based partitioning to minimize the amount of data that slots read from disk.

Once slot workers read their data from disk, BigQuery can automatically determine more optimal data sharding and quickly repartition data using BigQuery's in-memory shuffle service.

For more information, see Introduction to partitioned tables.

Clustering and sort keys

Redshift supports specifying table columns as either compound or interleaved sort keys. In BigQuery, you can specify compound sort keys by clustering your table. BigQuery clustered tables improve query performance because the table data is automatically sorted based on the contents of up to four columns specified in the table's schema. These columns are used to colocate related data. The order of the clustering columns you specify is important because it determines the sort order of the data.

Clustering can improve the performance of certain types of queries, such as queries that use filter clauses and queries that aggregate data. When data is written to a clustered table by a query job or a load job, BigQuery automatically sorts the data using the values in the clustering columns. These values are used to organize the data into multiple blocks in BigQuery storage. When you submit a query containing a clause that filters data based on the clustering columns, BigQuery uses the sorted blocks to eliminate scans of unnecessary data.

Similarly, when you submit a query that aggregates data based on the values in the clustering columns, performance is improved because the sorted blocks colocate rows with similar values.

Use clustering in the following circumstances:

- Compound sort keys are configured in your Redshift tables.
- Filtering or aggregation is configured against particular columns in your queries.

When you use clustering and partitioning together, your data can be partitioned by a date, timestamp, or integer column and then clustered on a different set of columns (up to four total clustered columns). In this case, data in each partition is clustered based on the values of the clustering columns.

When you specify sort keys in tables in Redshift, depending on the load on the system, Redshift automatically initiates the sort using your own cluster's compute capacity. You may even need to manually run the VACUUM command if you want to fully sort your table data as

soon as possible, for example, after a large data load. BigQuery automatically handles this sorting for you and does not use your allocated BigQuery slots therefore not affecting the performance of any of your queries.

For more information about working with clustered tables, see the Introduction to clustered tables.

Distribution keys

Redshift uses distribution keys to optimize the location of data blocks to execute its queries. BigQuery does not use distribution keys because it automatically determines and adds stages in a query plan (while the query is running) to improve data distribution throughout query workers.

External sources

If you use Redshift Spectrum to query data on Amazon S3, you can similarly use BigQuery's external data source feature to query data directly from files on Cloud Storage.

In addition to querying data in Cloud Storage, BigQuery offers federated query functions for querying directly from the following products:

- Cloud SQL (fully managed MySQL or PostgreSQL)
- Cloud Bigtable (fully managed NoSQL)
- Drive (CSV, JSON, Avro, Sheets)

Data locality

You can create your BigQuery datasets in both regional and multi-regional locations, whereas Redshift only offers regional locations. BigQuery determines the location to run your load, query, or export jobs based on the datasets referenced in the request. Refer to the BigQuery location considerations for tips on working with regional and multi-regional datasets.

Data type mapping in BigQuery

Redshift data types differ from BigQuery data types. For more details on BigQuery data types, refer to the official documentation.

BigQuery also supports the following data types, which do not have a direct Redshift analog.

- ARRAY
- BYTES
- GEOGRAPHY
- TIME
- STRU

SQL comparison

BigQuery standard SQL supports compliance with the SQL 2011 standard and has extensions that support querying nested and repeated data. Redshift SQL is based on PostgreSQL but has several differences which are detailed in the Redshift documentation. For a detailed comparison between Redshift and BigQuery SQL syntax and functions, see the Redshift to BigQuery SQL translation reference.

Postmigration

Reporting and analysis

Now that you've migrated your data warehouse into BigQuery, you can access a flexible suite of business intelligence (BI) solutions for reporting and analysis. Refer to the Reporting and analysis guide for more information on how you can use these solutions with BigQuery to get compelling insights from your data.

Performance optimization

Since you've migrated scripts that weren't designed with BigQuery in mind, you can opt to implement techniques for optimizing query performance in BigQuery. Refer to the Performance optimization guide for more information on how to implement these optional improvements to your BigQuery performance.