PROFESSIONAL SERVICES

Oracle® Database to BigQuery migration guide

Google Cloud

About this document 4

Introduction 5

Premigration 5

Migration scope and phases 5

Prepare and discover 5

Understanding your as-is data warehouse ecosystem 6

Assess and plan 7

Sample migration process 7

BigQuery capacity planning 7

Security in Google Cloud 8

Identity and access management 8

Row-level security 9

Full-disk encryption 11

Data masking and redaction 11

How is BigQuery different? 11

System architecture 11

Data and storage architecture 12

Query execution and performance 13

Agile analytics 13

High availability, backups, and disaster recovery 14

Caching 14

Connections 15

Pricing and licensing 15

Labeling 15

Monitoring and audit logging 15

Maintenance, upgrades, and versions 16

Workloads 17

Parameters and settings 17

Limits and quotas 17

BigQuery provisioning 17

Schema migration 18

Data structures and types 18

Oracle data types and BigQuery mappings 18

Indexes 18

Views 18

Materialized views 18

Table partitioning 19
Clustering 19
Temporary tables 20
External tables 20
Data modeling 20
Row format versus column format and server limits versus serverless 20
Denormalization 21
Techniques to flatten your existing schema 21
Example of flattening a star schema 22
Nested and repeated fields 25
Surrogate keys 26
Keeping track of changes and history 26
Role-specific and user-specific views 27

Data migration 27
Migration activities 27
Initial load 27
Constraints for initial load 28
Initial data transfer (batch) 28
Change Data Capture (CDC) and streaming ingestion from Oracle to BigQuery 28
Log-based CDC 29
SQL-based CDC 29
Triggers 30
ETL/ELT migration: Data and schema migration tools 30
Lift and shift approach 30
Re-architecting ETL/ELT platform 30
Cloud Data Fusion 30
Dataflow 32
Cloud Composer 32
Dataprep by Trifacta 36
Dataproc 36
Partner tools for data migration 37
Business intelligence (BI) tool migration 37
Query (SQL) translation 37

Migrating Oracle Database options 38
Oracle Advanced Analytics option 38
Oracle R Enterprise 38
Oracle Data Mining 38

Spatial and Graph option 39
Spatial 39
Graph 39
Oracle Application Express 39
Release notes **39**

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.

Highlights	
Purpose	The migration from Oracle Database or Exadata to BigQuery in Google Cloud.
Intended audience	Enterprise architects, DBAs, application developers, IT security.
Key assumptions	That the audience is an internal or external representative who: ● has been trained on basic Google Cloud products. ● is familiar with BigQuery and its concepts. ● is running OLAP workloads on Oracle Database or Exadata with intent to migrate those workloads to Google Cloud.

Objectives of this guide

1. Provide high-level guidance to organizations migrating from Oracle to BigQuery.
2. Show you new capabilities that are available to your organization through BigQuery, not map features one-to-one with Oracle.
3. Show ways organizations can rethink their existing data model and their extract, transform, and load (ETL), and extract, load, and transform (ELT) processes to get the most out of BigQuery.
4. Share advice and best practices for using BigQuery and other Google Cloud products.
5. Provide guidance on the topic of change management, which is often neglected.

Nonobjectives of this guide

1. Provide detailed instructions for all activities required to migrate from Oracle to BigQuery.
2. Present solutions for every use case of the Oracle platform in BigQuery.
3. Replace the 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 review the documentation before or while you read this guide.

Introduction

This document is part of the enterprise data warehouse (EDW) migration initiative. It focuses on technical differences between Oracle Database and BigQuery and approaches to migrating from Oracle to BigQuery. Most of the concepts described here can apply to Exadata, ExaCC, and Oracle Autonomous Data Warehouse as they use compatible Oracle Database software with optimizations and with smart and managed storage and infrastructure.

The document does not cover detailed Oracle RDBMS, Google Cloud, or BigQuery features and concepts. For more information, see Google Cloud documentation and Google Cloud solutions.

Premigration

There are many factors to consider depending on the complexity of the current environment. Scoping and planning are crucial to a successful migration.

Migration scope and phases

Data warehouse migrations can be complex, with many internal and external dependencies. Your migration plan should include defining your current and target architecture, identifying critical steps, and considering iterative approaches such as migrating one data mart at a time.

Prepare and discover

Identify one data mart to migrate. The ideal data mart would be:

- Big enough to show the power of BigQuery (50 GB to 10 TB).
- As simple as possible:
    - Identify the most business-critical or long-running reports, depending only on this data mart (for example, the top 20 long-running critical reports).
    - Identify the necessary tables for those reports (for example, one fact table and six dimension tables).
    - Identify the staging tables and necessary ETL and ELT transformations and streaming pipelines to produce the final model.
    - Identify the BI tools and consumers for the reports.

Define the extent of the use case:

- Lift and shift: Preserve the data model (star schema or snowflake schema), with minor changes to data types and the data consumption layer.

- Data warehouse modernization: Rebuild and optimize the data model and consumption layer for BigQuery through a process such as denormalization, discussed later in this document.
- End-to-end: Current source > ingestion > staging > transformations > data model > queries and reporting.
- ETL and data warehouse: Current staging > transformations > data model.
- ETL, data warehouse, and reporting: Current staging > transformations > data model > queries and reporting.
- Converting the data model: Current normalized star or snowflake schema on Oracle; target denormalized table on BigQuery.
- Converting or rewriting the ETL/ELT pipelines or PL/SQL packages.
- Converting queries for reports.
- Converting existing BI reports and dashboards to query BigQuery.
- Streaming inserts and analytics.

Optionally, identify current issues and pain points that motivate your migration to BigQuery.

Understanding your as-is data warehouse ecosystem

Every data warehouse has different characteristics depending on the industry, enterprise, and business needs. There are four high-level layers within every data warehouse ecosystem. It is crucial to identify components in each layer in the current environment.

Source systems

- On-premises databases
- Cloud databases
- CDC tools
- Other data sources (social media, IoT, REST)

Ingestion, processing, or transformation layer

- ETL/ELT tools
- Job orchestration
- Streaming tools
- Throughput
- Velocity, volume
- Data freshness (daily/hourly)
- Performance requirements (streaming latency, ETL/ELT windows)

Data warehouse

- Data marts (sales, HR, finance, marketing)
- Data models
- Star or snowflake
- Tables, materialized views (count, size)
- Staging area
- Transformations (ELT) (PL/SQL packages, DML, scheduled jobs)
- ILM jobs
- Partitioning
- Other workloads (OLTP, mixed) (Advanced Analytics, Spatial and Graph)
- Disaster recovery

Presentation

- Data visualization
- Data labs
- BI tools
- Applications

Assess and plan

Start by scoping a proof of concept (PoC) or pilot with one data mart. Following are some critical points for scoping a PoC for data warehouse migration.

Define the goal and success criteria for the PoC or pilot:

- Total cost of ownership (TCO) for data warehouse infrastructure (for example, compare TCO over four years, including storage, processing, hardware, licensing, and other dependencies like disaster recovery and backup).
- Performance improvements on queries and ETLs.
- Functional solutions or replacements and tests.
- Other benefits like scalability, maintenance, administration, high availability, backups, upgrades, performance tuning, machine learning, and encryption.

Sample migration process

If the data warehouse system is complex with many data marts, it might not be safe and easy to migrate the whole ecosystem. An iterative approach like the following might be better for complex data warehouses:

- **Phase 1:** Prepare and discover (determine the current and target architectures for one data mart).
- **Phase 2:** Assess and plan (implement pilot for one data mart, define measures of success, and plan your production migration).
- **Phase 3:** Execute (dual run, moving user access to Google Cloud gradually and retiring migrated data mart from previous environment).

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 need (instead of bytes processed). By default, Google Cloud projects are limited to a maximum of 2,000 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 can 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 any questions regarding your current plan and options listed above, contact your sales representative to take advantage of BigQuery Reservations.

Security in Google Cloud

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

Identity and access management

Oracle provides users, privileges, roles, and profiles to manage access to resources.

BigQuery uses Identity and Access Management (IAM) to manage access to resources and provides centralized access management to resources and actions. The types of resources available in BigQuery are organizations, projects, datasets, tables, 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 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.

Row-level security

Oracle Label Security (OLS) allows the restriction of data access on a row-by-row basis. A typical use case for row-level security is restricting a sales person's access to the accounts they manage. By implementing row-level security, you gain fine-grained access control.

Although BigQuery does not have an explicit row-level security module, you can achieve the same functionality by using authorized views. Authorized views let you permit users to query a view without having read access to the underlying tables. To create row-level security over a table, you create an authorized view that returns only the rows appropriate for that security category.

If you want to give different users access to different rows in your table, without having to create an individual view for each, there are a few options. These options all make use of the `SESSION_USER()` function in BigQuery, which returns the email address of the currently running user. In this example, `example@google.com` represents that email address.

In practice, you'll want to specify a group. Your schema looks like `{customer:string, id:integer, allowed_group: string}`, and anyone in `allowed_group` can see your table. The `allowed_group` members are represented in another table that contains your group mappings. That table looks like this: `{group:string, user_name:string}`. The rows might look like this:

```
{engineers, example@google.com}
{engineers, some_engineer@google.com}
{administrators, some_admin@google.com}
```

```
{sales, some_salesperson@google.com}
...
```

After creating the group mapping table, you can update the view definition defined previously as follows (where `private.access_control` holds the group mapping):

```
#standardSQL
SELECT c.customer, c.id
FROM private.customers c
INNER JOIN (
    SELECT DISTINCT group
    FROM private.access_control
    WHERE user_name = SESSION_USER()) g
ON c.allowed_group = g.group
```

If multiple access levels are needed depending on a role, the sample query for the authorized view might look like the following:

```
SELECT c.customer, c.id
FROM private.customers c
JOIN private.access_control g
ON g.user_name=SESSION_USER()
WHERE (CASE WHEN g.role='Representative' THEN c.representative =
g.user_name
        WHEN g.role='Unit Manager' THEN c.unit = g.unit
        WHEN g.role='Director' THEN c.department = g.department
        ELSE 1=0
        END
  )
```

In this way, you can manage the access control group mapping (`private.access_control`) separately from the data table (`private.customers`). You can also add the ability for groups to contain other groups. You can get this by doing a more complex join to expand the groups in the access control table. Consider saving the results each time you query the main table.

There are other options like adding another column to your table for the user who is allowed to see the row. For example, the schema `{customer:string, id:integer}` would become `{customer:string, id:integer, allowed_viewer: string}`. Then you can define a view as follows:

```
#standardSQL
SELECT customer, id
FROM private.customers
WHERE allowed_viewer = SESSION_USER()
```

The user querying the view can see only those rows where the current user (`example@google.com`) was the value in the `allowed_viewer` column. Disadvantages to this option are that you can grant access to only one user at a time, and the access permission exists within the row itself and is therefore difficult to update. To avoid these drawbacks, you can make the `allowed_viewer` column a repeated field. This lets you provide a list of users for each row. This approach is also difficult to update and requires you to keep track of which users should have access to which rows.

Full-disk encryption

Oracle offers Transparent Data Encryption (TDE) and network encryption for data-at-rest and in-transit encryption. TDE requires the Advanced Security option, which is licensed separately.

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. For more information about encryption at Google Cloud, see whitepapers on data-at-rest encryption and encryption-in-transit.

Data masking and redaction

Oracle uses Data Masking in Real Application Testing and Data Redaction, which lets you mask (redact) data that is returned from queries issued by applications.

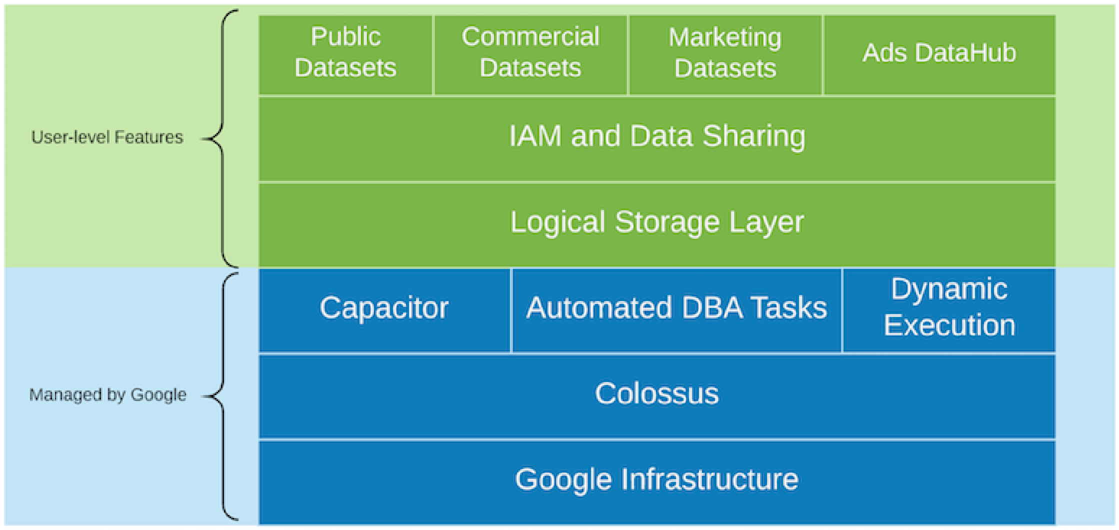
You can use the Google Data Loss Prevention (DLP) API to identify and redact sensitive personally identifiable information (PII) on BigQuery.

How is BigQuery different?

Note: This section does not cover all BigQuery features and tools but concentrates on key differences between BigQuery and Oracle Database. These highlights help you identify migration hurdles and plan for the changes required.

System architecture

One of the main differences between Oracle Database and BigQuery is that BigQuery is a serverless cloud EDW with separate storage and compute layers that can scale based on the needs of the query. Given the nature of BigQuery's serverless offering, users are not limited by hardware decisions; instead users can request more resources for their queries and users through reservations. BigQuery also does not require configuration on the underlying software and infrastructure such as OS, network, and storage systems including scaling and high-availability. BigQuery takes care of scalability, management, and administrative operations. The following diagram illustrates BigQuery's storage hierarchy.



Knowledge of the underlying storage and query processing architecture such as separation between storage (Colossus) and query execution (Dremel) and how Google Cloud allocates resources (Borg) can be good for understanding behavioral differences and optimizing query performance and cost effectiveness. For details, see the reference system architectures for BigQuery, Oracle Database, and Exadata.

Data and storage architecture

The data and storage structure is an important part of any data analytics system because it affects query performance, cost, scalability, and efficiency.

BigQuery decouples data storage and compute and stores data in Colossus, in which data is compressed and stored in a columnar format called Capacitor.

BigQuery operates directly on compressed data without decompressing by using Capacitor. BigQuery provides datasets as the highest-level abstraction to organize access to tables as shown in the diagram above. Schemas and labels can be used for further organization of tables. BigQuery offers partitioning to improve query performance and costs and to manage information lifecycle. Storage resources are allocated as you consume them and deallocated as you remove data or drop tables.

Oracle Database stores data in row format using Oracle block format organized in segments. Schemas (owned by users) are used to organize tables and other database objects. As of Oracle Database 12c, multitenant is used to create pluggable databases within one database instance for further isolation. Partitioning can be used to improve query performance and information lifecycle operations. Oracle offers several storage options for standalone and Real Application Clusters (RAC) databases such as ASM, an OS file system, and a cluster file system.

Exadata provides optimized storage infrastructure in storage cell servers and allows Oracle Database servers to access this data transparently by utilizing ASM. Exadata offers Hybrid Columnar Compression (HCC) options so that users can compress tables and partitions. Oracle requires preprovisioned storage capacity, careful sizing, and autoincrement configurations on segments, datafiles, and tablespaces.

Query execution and performance

BigQuery manages performance and scales on the query level to maximize performance for the cost. BigQuery uses many optimizations, for example:

- In-memory query execution
- Multilevel tree architecture based on Dremel execution engine
- Automatic storage optimization within Capacitor
- 1 petabit per second total bisection bandwidth with Jupiter
- Autoscaling resource management to provide fast petabyte-scale queries

BigQuery gathers column statistics while loading the data and includes diagnostic query plan and timing information. Query resources are allocated according to query type and complexity. Each query uses some number of slots, which are units of computation that comprise a certain amount of CPU and RAM.

Oracle provides data statistics gathering jobs. The database optimizer uses statistics to provide optimal execution plans. Indexes might be needed for fast row lookups and join operations. Oracle also provides an in-memory column store for in-memory analytics. Exadata provides several performance improvements such as cell smart scan, storage indexes, flash cache, and InfiniBand connections between storage servers and database servers. Real Application Clusters (RAC) can be used for achieving server high availability and scaling database CPU-intensive applications using the same underlying storage.

Optimizing query performance with Oracle Database requires careful consideration of these options and database parameters. Oracle provides several tools such as Active Session History (ASH), Automatic Database Diagnostic Monitor (ADDM), Automatic Workload Repository (AWR) reports, SQL monitoring and Tuning Advisor, and Undo and Memory Tuning Advisors for performance tuning.

Agile analytics

In BigQuery, you can enable different projects, users, and groups to query datasets in different projects. Separation of query execution lets autonomous teams work within their projects without affecting other users and projects by separating slot quotas and querying billing from other projects and the projects that host the datasets.

High availability, backups, and disaster recovery

Oracle provides Data Guard as a disaster recovery and database replication solution. Real Application Clusters (RAC) can be configured for server availability. Recovery Manager (RMAN) backups can be configured for database and archivelog backups and also used for restore and recovery operations. The Flashback database feature can be used for database flashbacks to rewind the database to a specific point in time. Undo tablespace holds table snapshots. It is possible to query old snapshots with the flashback query and "as of" query clauses depending on the DML/DDL operations done previously and the undo retention settings. In Oracle, the whole integrity of the database should be managed within tablespaces that depend on the system metadata, undo, and corresponding tablespaces, because strong consistency is important for Oracle backup, and recovery procedures should include full master data. You can schedule exports on the table schema level if point-in-time recovery is not needed in Oracle.

BigQuery is fully managed and different from traditional database systems in its complete backup functionality. You don't need to consider server, storage failures, system bugs, and physical data corruptions. BigQuery replicates data across different data centers depending on the dataset location to maximize reliability and availability. BigQuery multi-region functionality replicates data across different regions and tolerates loss of a complete region. BigQuery single-region functionality replicates data across different zones within the same region. BigQuery lets you query historical snapshots of tables up to seven days and restore deleted tables within two days by using table decorators. You can copy a deleted table (in order to restore it) by using the snapshot syntax (`dataset.table@timestamp`). You can export data from BigQuery tables for additional backup needs such as to recover from accidental user operations. Proven backup strategy and schedules used for existing data warehouse (DWH) systems can be used for backups.

Batch operations and the snapshotting technique allow different backup strategies for BigQuery. You don't need to export unchanged tables and partitions frequently. One export backup of the partition or table is enough after the load or ETL operation finishes. To reduce backup cost, you can store export files in Cloud Storage nearline or coldline storage and define a lifecycle policy to delete files after a certain amount of time depending on the data retention requirements.

Caching

BigQuery offers per-user cache, and if data doesn't change, results of queries are cached for approximately 24 hours. If the results are retrieved from the cache, the query costs nothing.

Oracle offers several caches for data and query results such as buffer cache, result cache, Exadata Flash Cache, and in-memory column store.

Connections

BigQuery handles connection management and does not require you to do any server-side configuration. BigQuery provides JDBC and ODBC drivers. You can use the Cloud Console or the bq command-line tool for interactive querying. You can use REST APIs and client libraries to programmatically interact with BigQuery. You can connect Google Sheets directly with BigQuery, and there is also a BigQuery connector for Excel. If you are looking for a desktop client, there are free tools like DBeaver.

Oracle Database provides listeners, services, service handlers, several configuration and tuning parameters, and shared and dedicated servers to handle database connections. Oracle provides JDBC, JDBC Thin, ODBC drivers, Oracle Client, and TNS connections. Scan listeners, scan IP addresses, and scan-name are needed for RAC configurations.

Pricing and licensing

Oracle requires license and support fees based on the core counts for Database editions and Database options such as RAC, multitenant, Active Data Guard, partitioning, in-memory, Real Application Testing, GoldenGate, and Spatial and Graph.

BigQuery offers flexible pricing options based on storage, query, and streaming inserts usage. BigQuery offers flat-rate pricing for customers who need predictable cost and slot capacity in specific regions. Slots that are used for streaming inserts and loads are not counted on project slot capacity and flat-rate. Refer to the BigQuery capacity planning section for help deciding how many slots you'll purchase for your data warehouse.

BigQuery also automatically cuts storage costs in half for unmodified data stored for more than 90 days.

Labeling

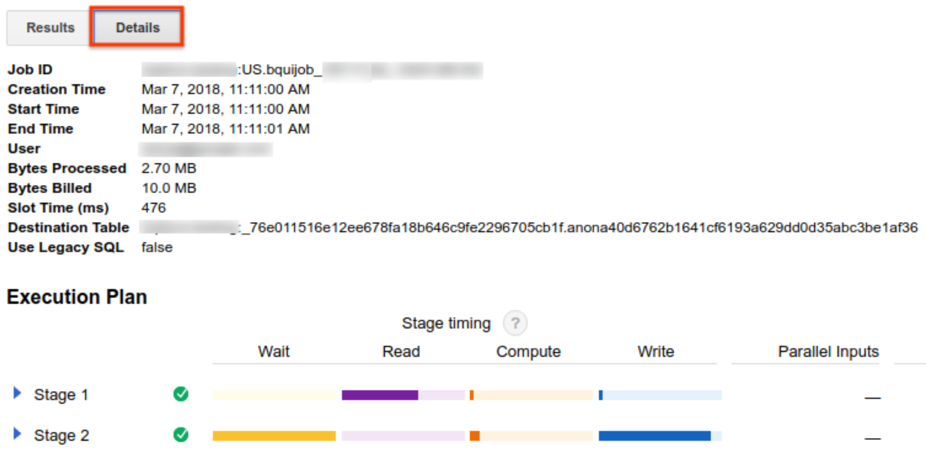
BigQuery datasets, tables, and views can be labeled with key-value pairs. Labels can be used for differentiating storage costs and internal chargebacks.

Monitoring and audit logging

Oracle provides different levels and kinds of database auditing options and audit vault and database firewall features, which are licensed separately. Oracle provides Enterprise Manager for database monitoring.

For BigQuery, Cloud Audit Logs is used for both data access logs and audit logs, which are enabled by default. The data access logs are available for 30 days, and the other system events and admin activity logs are available for 400 days. If you need longer retention, you can export logs to BigQuery, Cloud Storage, or Pub/Sub as described here. If integration with an existing incident monitoring tool is needed, Pub/Sub can be used for exports, and custom development should be done on the existing tool to read logs from Pub/Sub.

Audit logs include all the API calls, query statements, and job statuses. You can use Cloud Monitoring to monitor slot allocation, bytes scanned in queries and stored, and other BigQuery metrics. BigQuery query plan and timeline can be used to analyze query stages and performance.



Job ID	[redacted]:US.bquijob_[redacted]
Creation Time	Mar 7, 2018, 11:11:00 AM
Start Time	Mar 7, 2018, 11:11:00 AM
End Time	Mar 7, 2018, 11:11:01 AM
User	[redacted]
Bytes Processed	2.70 MB
Bytes Billed	10.0 MB
Slot Time (ms)	476
Destination Table	[redacted]:_76e011516e12ee678fa18b646c9fe2296705cb1f.anona40d6762b1641cf6193a629dd0d35abc3be1af36
Use Legacy SQL	false

Execution Plan

	Stage timing					Rows	
	Wait	Read	Compute	Write	Parallel Inputs	Input	Output
Stage 1 ✓					—	34.4 K	10 (951 B)
Stage 2 ✓					—	10	10 (831 B)

You can follow the error messages table for troubleshooting query job and API errors. To distinguish slot allocations per query or job, you can use this utility, which is beneficial for customers that use flat-rate pricing and have many projects distributed across several teams.

Maintenance, upgrades, and versions

BigQuery is a fully managed service and does not require you to do any maintenance or upgrades. BigQuery does not offer different versions. Upgrades are continuous and don't require downtime or hinder system performance. For more information, see Release notes.

Oracle Database and Exadata require users to do database and underlying infrastructure-level patching, upgrades, and maintenance. There are many versions of Oracle Database and a new major version is planned to be released every year. Although new versions are backward-compatible, query performance, context, and features can change.

There can be applications that require specific versions such as 10g, 11g, or 12c. Careful planning and testing are needed for major database upgrades. Migration from different versions might include different technical conversion needs on query clauses and database objects.

Workloads

Oracle Exadata supports mixed workloads including OLTP workloads. BigQuery is designed for analytics and is not designed to handle OLTP workloads. OLTP workloads that use the same Oracle database should be migrated into Cloud SQL, Cloud Spanner, or Firestore in Google Cloud. Oracle Database offers additional options such as Advanced Analytics and Spatial and Graph. These workloads might need to be rewritten for migration to BigQuery. For more information, see the Migrating Oracle Database options section in this document.

Parameters and settings

Oracle offers and requires many parameters to be configured and tuned on the OS, Database, RAC, ASM, and Listener levels for different workloads and applications. BigQuery is a fully managed service and does not require you to configure any initialization parameters.

Limits and quotas

Oracle Database has hard and soft limits based on infrastructure, hardware capacity, parameters, software versions, and licensing. BigQuery has quotas and limits on specific actions and objects.

BigQuery provisioning

BigQuery is a platform as a service (PaaS) and a Cloud massively parallel processing data warehouse. Its capacity scales up and down without any intervention from the user as Google manages the backend. As a result, unlike many RDBMS systems, BigQuery doesn't need you to provision resources before use. BigQuery allocates storage and query resources dynamically based on your usage patterns. Storage resources are allocated as you consume them and deallocated as you remove data or drop tables. Query resources are allocated according to query type and complexity. Each query uses slots. An eventual fairness scheduler is used, so there might be short periods where some queries get a higher share of slots, but the scheduler eventually corrects this.

In traditional VM terms, BigQuery gives you the equivalent of both:

- Per-second billing
- Per-second scaling

To accomplish this task, BigQuery does the following:

- Keeps vast resources deployed under the hood to avoid having to rapidly scale.
- Uses multitenant resources to instantly allocate large chunks for seconds at a time.
- Efficiently allocates resources across users with economies of scale.
- Charges you only for the jobs you run, rather than for deployed resources, so you pay for resources you use.

For more information about pricing, see Understanding BigQuery's rapid scaling and simple pricing.

Schema migration

In this chapter, you can find data schema considerations for Oracle to BigQuery migration, such as data structures and types, performance-related options, and data modeling considerations and best practices.

Data structures and types

Oracle data types and BigQuery mappings

Oracle data types differ from BigQuery data types. For more information about BigQuery data types, refer to the official documentation.

For a detailed comparison between Oracle and BigQuery data types, see the Oracle to BigQuery SQL translation reference.

Indexes

In many analytical workloads, columnar tables are used instead of row stores. This greatly increases the column-based operations and eliminates the use of indexes for batch analytics. BigQuery also stores data in a columnar format, so indexes are not needed in BigQuery. If the analytics workload requires a single, small set of row-based access, Cloud Bigtable can be a better alternative. If a workload requires transaction processing with strong relational consistencies, Cloud Spanner or Cloud SQL can be better alternatives.

To summarize, no indexes are needed and offered in BigQuery for batch analytics. Partitioning or clustering can be used; these concepts are further explained in this section. For more information about how to tune and improve query performance in BigQuery, see Introduction to optimizing query performance.

Views

Similar to Oracle, BigQuery allows creating custom views. However, views in BigQuery do not support DML statements. This behavior might change in future releases. See the documentation to get the most up-to-date information regarding this functionality.

Materialized views

Materialized views are commonly used to improve report rendering time in write-once, read-many types of reports and workloads.

Materialized views are offered in Oracle to increase view performances by simply creating and maintaining a table to hold the query result dataset. There are two ways to refresh materialized views in Oracle: on-commit and on-demand.

Materialized view functionality is also available in BigQuery (in beta as of August 2020). BigQuery leverages precomputed results from materialized views and whenever possible reads only delta changes from the base table to compute up-to-date results.

Caching functionalities in Google Data Studio or other modern BI tools can also improve performance and eliminate the need to rerun the same query, saving costs.

Table partitioning

Table partitioning is widely used in Oracle data warehouses. In contrast to Oracle, BigQuery does not support hierarchical partitioning.

BigQuery implements three types of table partitioning that allow queries to specify predicate filters based on the partitioning column to reduce the amount of data scanned.

- 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.

For more information about limits and quotas applied to partitioned tables in BigQuery, see Introduction to partitioned tables.

If the above BigQuery restrictions affect the functionality of the migrated database, consider using sharding instead of partitioning.

Further, BigQuery does not support `EXCHANGE PARTITION`, `SPLIT PARTITION`, or converting a nonpartitioned table to a partitioned one.

Clustering

Clustering helps efficiently organize and retrieve data stored in multiple columns that is often accessed together. However, Oracle and BigQuery have different circumstances in which clustering works best. In BigQuery, if a table is commonly filtered and aggregated with specific columns, clustering can be used. A clustered table in BigQuery might require reclustering if it was often modified and became partially sorted over time. Clustering can be considered for migrating list-partitioned or index-organized tables from Oracle.

Temporary tables

Temporary tables are often used in Oracle ETL pipelines. A temporary table holds data during a user session. This data is automatically deleted at the end of the session.

BigQuery uses temporary tables to cache query results that aren't written to a permanent table. After a query finishes, the temporary tables exist for up to 24 hours. The tables are created in a special dataset and named randomly. You can also create temporary tables for your own use. For more information, see Temporary tables.

External tables

Similar to Oracle, BigQuery lets you query external data sources. BigQuery supports querying data directly from Bigtable, Cloud Storage, Google Drive, and Cloud SQL.

Data modeling

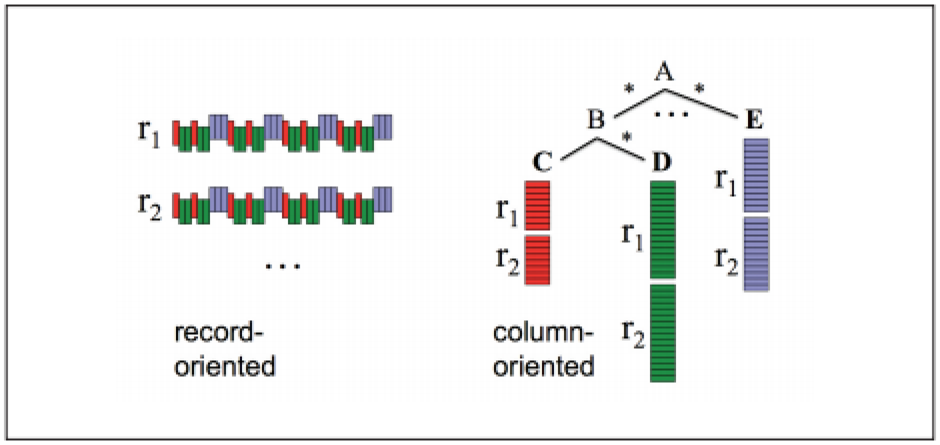
Star or snowflake data models can be efficient for analytics storage and are commonly used for data warehouses on Oracle Exadata.

Denormalized tables eliminate expensive join operations and in most cases provide better performance for analytics in BigQuery. Star and snowflake data models are also supported by BigQuery join performance. For more data warehouse design details on BigQuery, see Designing schema.

Row format versus column format and server limits versus serverless

Oracle uses a row format in which the table row is stored in data blocks, so unneeded columns are fetched within the block for the analytic queries based on filtering and aggregation of specific columns.

Oracle has a shared-everything architecture with fixed hardware resource dependencies such as memory and storage assigned to the server. These are the two main forces underlying many data modeling techniques that evolved to improve efficiency of storage and performance of analytic queries. Star and snowflake schemas and data vault modeling are some of these.



Columnar storage of Dremel

BigQuery uses column format, as in the preceding diagram, to store data and does not have fixed storage and memory limits. This architecture lets developers further denormalize and design schemas based on reads and business needs, reducing complexity and improving flexibility, scalability, and performance.

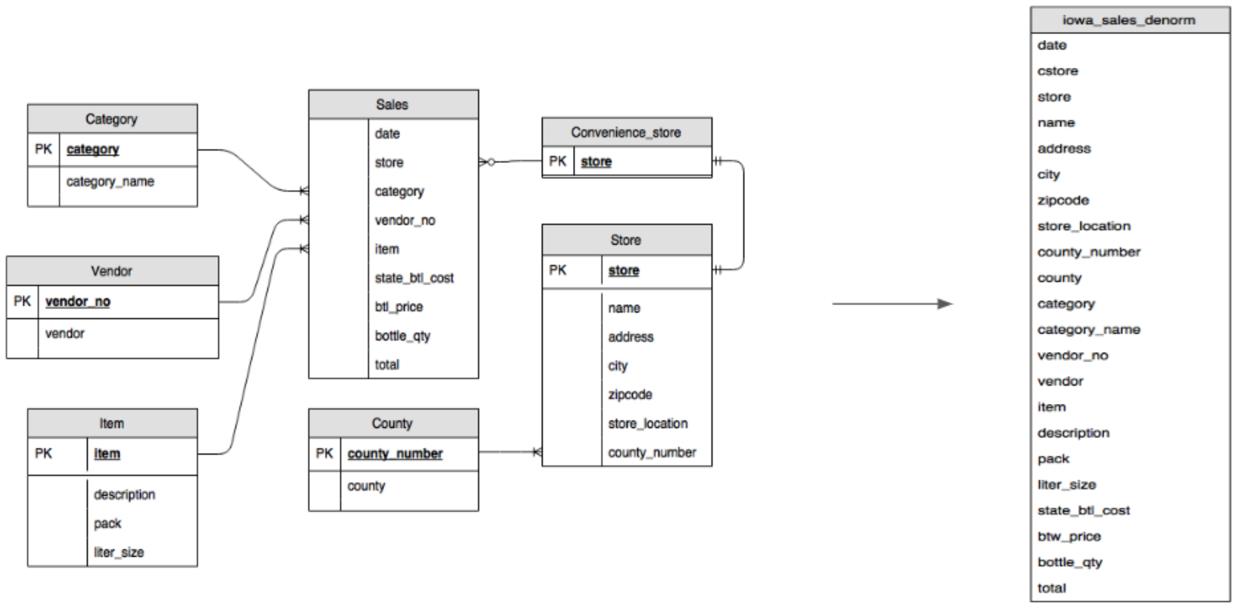
Denormalization

One of the main goals of relational database normalization is to reduce data redundancy. While this model is best suited for a relational database that uses row format, data denormalization is preferable for columnar databases. For more information about advantages of denormalizing data and other query optimization strategies in BigQuery, see Denormalization.

Techniques to flatten your existing schema

BigQuery technology leverages a combination of columnar data access and processing, in-memory storage, and distributed processing to provide quality query performance.

When designing a BigQuery DWH schema, creating a fact table in a flat table structure (consolidating all the dimension tables into a single record in the fact table) is better for storage utilization than using multiple DWH dimensions tables. On top of less storage utilization, having a flat table in BigQuery leads to less `JOIN` usage. The following diagram illustrates an example of flattening your schema.



Example of flattening a star schema

Figure 1 shows a fictional sales management database that includes four tables:

- Orders/sales table (fact table)
- Employee table
- Location table
- Customer table

The primary key for the sales table is the `OrderNum`, which also contains foreign keys to the other three tables.

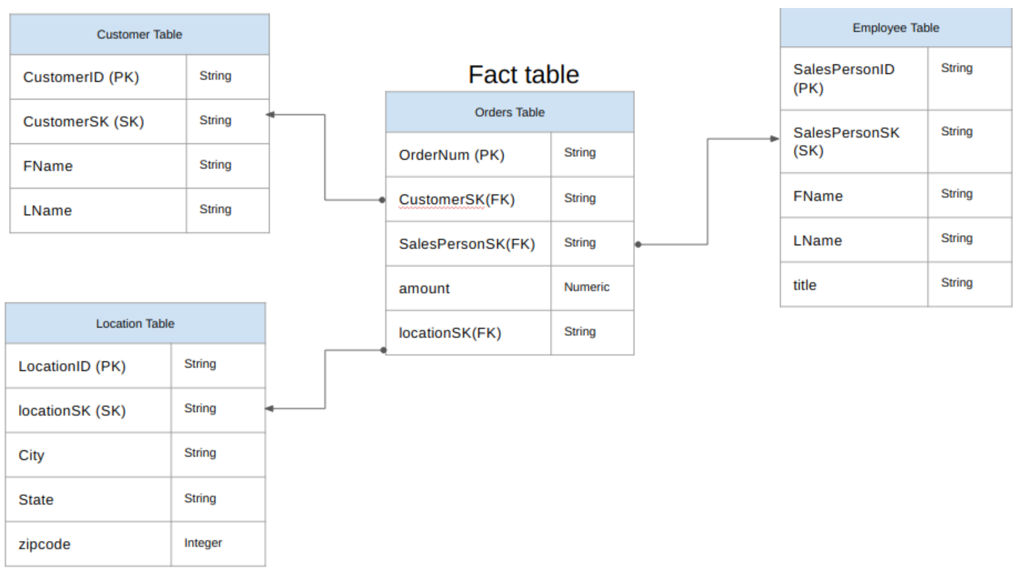


Figure 1: Sample sales data in a star schema

Sample data

Orders/fact table content

OrderNum	CustomerID	SalesPersonID	amount	Location
O-1	1234	12	234.22	18
O-2	4567	1	192.10	27
O-3		12	14.66	18
O-4	4567	4	182.00	26

Employee table content

SalesPersonID	FName	LName	title
1	Alex	Smith	Sales Associate
4	Lisa	Doe	Sales Associate
12	John	Doe	Sales Associate

Customer table content

CustomerID	FName	LNAME
1234	Amanda	Lee
4567	Matt	Ryan

Location table content

Location	city	state	zipcode
18	Bronx	NY	10452
26	Mountain View	CA	90210
27	Chicago	IL	60613

Query to flatten the data using LEFT OUTER JOINs

```
#standardSQL
INSERT INTO flattened
SELECT
    orders.ordernum,
    orders.customerID,
    customer.fname,
    customer.lname,
    orders.salespersonID,
    employee.fname,
    employee.lname,
    employee.title,
    orders.amount,
    orders.location,
    location.city,
    location.state,
    location.zipcode
FROM orders
LEFT OUTER JOIN customer
    ON customer.customerID = orders.customerID
LEFT OUTER JOIN employee
    ON employee.salespersonID = orders.salespersonID
LEFT OUTER JOIN location
    ON location.locationID = orders.locationID
```

Note: BigQuery processes JOINs from left to right (top to bottom). Placing JOINs that reduce the number of records as early as possible in the chain increases query efficiency by reducing response time.

▶ *Output of the flattened data*

OrderNum	CustomerID	FName	LName	SalesPersonID	FName	LName	amount	Location	city	state	zipcode
O-1	1234	Amanda	Lee	12	John	Doe	234.22	18	Bronx	NY	10452
O-2	4567	Matt	Ryan	1	Alex	Smith	192.10	27	Chicago	IL	60613
O-3				12	John	Doe	14.66	18	Bronx	NY	10452
O-4	4567	Matt	Ryan	4	Lisa	Doe	182.00	26	Mountain View	CA	90210

Nested and repeated fields

In order to design and create a DWH schema from a relational schema (for example, star and snowflake schemas holding dimension and fact tables), BigQuery presents the nested and repeated fields functionality. Therefore, relationships can be preserved in a similar way as a relational normalized (or partial normalized) DWH schema without performance impact. For more information, see performance best practices.

To better understand the implementation of nested and repeated fields, look at a simple relational schema of a `CUSTOMERS` table and `ORDER/SALES` table. They are two different tables, one for each entity, and the relationships are defined using a key such as a primary key and a foreign key as the link between the tables while querying using `JOIN`s. BigQuery nested and repeated fields let you preserve the same relationship between the entities in one single table. This can be implemented by having all the customer data, while the orders data is nested for each of the customers. For more information, see Specifying nested and repeated columns.

To convert the flat structure into a nested or repeated schema, nest the fields as follows:

- `CustomerID`, `FName`, `LName` nested into a new field called `Customer`.
- `SalesPersonID`, `FName`, `LName` nested into a new field called `Salesperson`.
- `LocationID`, `city`, `state`, `zip code` nested into a new field called `Location`.

Fields `OrderNum` and `amount` are not nested, as they represent unique elements.

You want to make your schema flexible enough to allow for every order to have more than one customer: a primary and a secondary. The customer field will accordingly be marked as repeated. The resulting schema is shown in Figure 2, which illustrates nested and repeated fields.

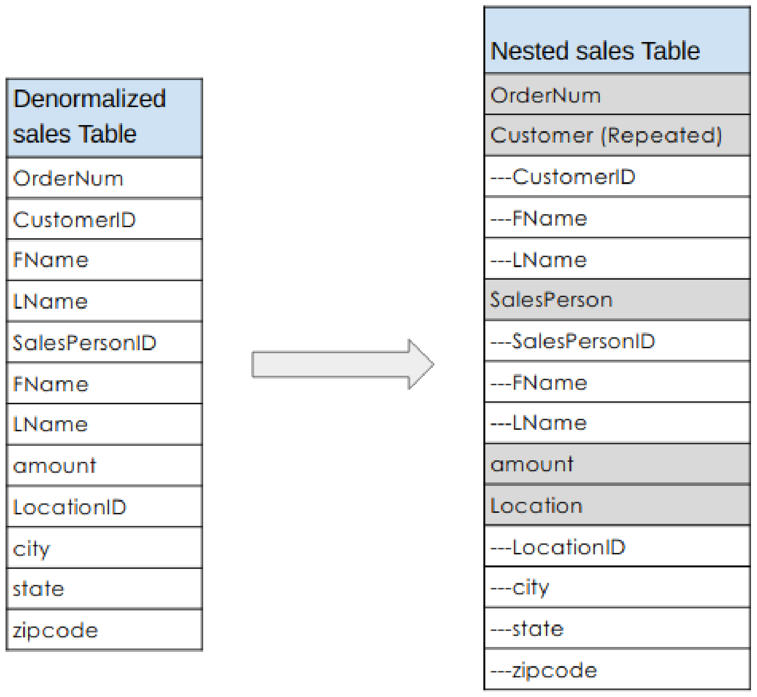


Figure 2: Logical representation of a nested structure

Note that in some cases, denormalization using nested and repeated fields does not lead to performance improvements. For more information about limitations and restrictions of nested and repeated fields, see Loading denormalized, nested, and repeated data.

Surrogate keys

It is common to identify rows with unique keys within the tables. Sequences are commonly used at Oracle databases to create these keys. In BigQuery, you can create surrogate keys by using `row_number` and `partition by` functions. For more information, see BigQuery and surrogate keys: a practical approach.

Keeping track of changes and history

When planning a BigQuery DWH migration, consider the concept of *slowly changing dimensions* (SCD). In general, the term SCD describes the process of making changes (DML operations) in the dimension tables. For best practices to handle SCD on BigQuery, see Slowly changing dimensions.

For several reasons, traditional data warehouses use different types for handling data changes and keeping historical data in slowly changing dimensions. These type usages are needed by the hardware limitations and efficiency requirements discussed earlier. Because the storage is much cheaper than compute and infinitely scalable, data redundancy and duplication is encouraged if it results in faster queries in BigQuery. You can use data snapshotting techniques in which the whole data is loaded into new daily partitions.

Role-specific and user-specific views

Use role-specific and user-specific views when different users belong to different teams and should see only records and results that they need. Row-based filtering based on user login can be done with a `session_user` clause on BigQuery. Add a `user_email` column to the underlying table, and create authorized views in other datasets by adding `user_email=session_user` in the `where` clause and granting user access only to those views.

Data migration

In this chapter, you can find the critical parts of any data migration from Oracle to BigQuery, including initial load, change data capture (CDC), and ETL/ELT tools and approaches.

Migration activities

It is recommended to perform migration in phases by identifying appropriate use cases for migration. For more information, see the Migration scope and phases section of this document. There are multiple tools and services available to migrate data from Oracle to Google Cloud. While this list is not exhaustive, it does provide a sense of a migration effort's size and scope.

- **Exporting data out of Oracle:** In this document, see the sections Initial load and Change data capture (CDC) and streaming ingestion from Oracle to BigQuery. Also see the section ETL/ELT migration, as ETL tools can be used for the initial load as well.
- **Data staging (in Cloud Storage):** Cloud Storage is the recommended landing place (staging area) for data exported from Oracle. Cloud Storage is designed for fast, flexible ingestion of structured or unstructured data.
- **ETL process:** In this document, see ETL/ELT migration.
- **Loading data directly into BigQuery:** Data can be pushed into BigQuery's data store either directly through a data load from Cloud Storage, through Dataflow, or through real-time streaming. Dataflow is used when data transformation is required.

Initial load

Migration of the initial data from the existing Oracle data warehouse to BigQuery might be different from the incremental ETL/ELT pipelines depending on the data size and the network bandwidth. The same ETL/ELT approach can be used if the data size is a couple of terabytes.

If the data is up to a few terabytes, dumping the data and using gsutil for the transfer can be much more efficient than using JdbcIO-like programmatic database extraction methodology because programmatic approaches might need much more granular performance tuning. If the data size is more than a few terabytes and stored in cloud or online storage (such as Amazon S3), consider Cloud Data Transfer Service. For large-scale transfers (especially transfers with limited network bandwidth), Transfer Appliance is a useful option.

Constraints for initial load

When planning for data migration, consider the following:

- **Oracle DWH data size:** The source schemas' size carries a significant weight on the chosen data transfer method, especially when the data size is large (terabytes and above). When the data size is relatively small, the data transfer process can be completed in fewer steps. Dealing with large-scale data sizes makes the overall process more complex.
- **Downtime:** Deciding whether downtime is an option for your migration to BigQuery is important. To reduce downtime, you can bulk load the steady historical data and have a CDC solution to catch up with changes that happen during the transfer process.
- **Pricing:** In some scenarios, you might need third-party integration tools (for example, ETL or replication tools) that require additional licenses.

Initial data transfer (batch)

Data transfer using a batch method indicates that the data would be exported consistently in a single process (for example, exporting the Oracle DWH schema data into CSV, Avro, or Parquet files or importing to Cloud Storage to create datasets on BigQuery). Note that all the ETL tools and concepts explained in this document's ETL/ELT migration section can be used for the initial load.

If you do not want to use an ETL/ELT tool for the initial load, you can write custom scripts to export data to files (CSV, Avro, or Parquet) and upload that data to Cloud Storage using gsutil, Cloud Data Transfer Service, or Transfer Appliance. (For more information about performance tuning large data transfers and transfer options, see Transferring your large data sets.) Then load data from Cloud Storage to BigQuery as described here.

Cloud Storage is ideal for handling the initial landing for data. Cloud Storage is a highly available and durable object storage service with no limitations on the number of files, and you pay only for the storage you use. The service is optimized to work with other Google Cloud services such as BigQuery and Dataflow.

Change Data Capture (CDC) and streaming ingestion from Oracle to BigQuery

There are several ways to capture the change data from Oracle. Each option has trade-offs, mainly in the performance impact on the source system, development and configuration requirements, and pricing and licensing.

Log-based CDC

- *Oracle GoldenGate for Big Data*
  Oracle GoldenGate is Oracle's recommended tool for extracting redo logs, and you can use GoldenGate for Big Data for streaming logs into BigQuery. GoldenGate requires per-CPU licensing, and you can see the list price here. If Oracle GoldenGate for Big Data is available (in case licenses have already been acquired), using GoldenGate can be a good choice to create data pipelines to transfer data (initial load) and then sync all data modification.

- *Oracle XStream*
  Oracle Database stores every commit in redo log files, and these redo files can be used for CDC. Oracle XStream Out is built on top of LogMiner and provided by third-party tools such as Debezium (as of version 0.8) or commercially using tools such as Alooma or Striim. Using XStream APIs requires purchasing a license for Oracle GoldenGate even if GoldenGate is not installed and used. XStream enables you to propagate Streams messages between Oracle databases and the outside world efficiently.

- *Oracle LogMiner*
  No special license is required for LogMiner. You can use the LogMiner option in the Debezium community connector. It is also available commercially using tools such as Attunity, Striim, or StreamSets. LogMiner might have some performance impact on a very active source database and should be used carefully in cases when the volume of changes (the size of the redo) is more than 10 GB per hour depending on the server's CPU, memory, and I/O capacity and utilization.

SQL-based CDC

This is the incremental ETL approach in which SQL queries continuously poll the source tables for any changes depending on a monotonically increasing key and a timestamp column that holds the last modified or inserted date. If there is no monotonically increasing key, using the timestamp column (modified date) with a small precision (seconds) can cause duplicate records or missed data depending on the volume and comparison operator, such as > or >=.

To overcome such issues, you can use higher precision in timestamp columns such as six fractional digits (microseconds, which is the maximum supported precision in BigQuery), or you can add deduplication tasks in your ETL/ELT pipeline, depending on the business keys and data characteristics.

There should be an index on the key or timestamp column for better extract performance and less impact on the source database. Delete operations are a challenge for this methodology because they should be handled in the source application in a soft delete way, such as putting a deleted flag and updating `last_modified_date`. An alternative solution can be logging these operations in another table using a trigger, described later in this section.

Triggers

Database triggers can be created on source tables to log changes into shadow journal tables. Journal tables can hold entire rows to keep track of every column change, or they can only keep the primary key with the operation type (insert, update, or delete). Then changed data can be captured with an SQL-based approach described in this document under SQL-based CDC. Using triggers can affect the transaction performance and double the single-row DML operation latency if a full row is stored. Storing only the primary key can reduce this overhead, but in that case, a `JOIN` operation with the original table is required in the SQL-based extraction, which will miss the intermediate changes.

ETL/ELT migration: Data and schema migration tools

There are many possibilities for handling ETL/ELT on Google Cloud. Technical guidance on specific ETL workload conversions is not in the scope of this document. You can consider a lift and shift approach or re-architect your data integration platform depending on constraints such as cost and time. For more information about how to migrate your data pipelines to Google Cloud and many other migration concepts, see Migrating data warehouses to BigQuery: Data pipelines.

Lift and shift approach

If your existing platform supports BigQuery and you want to continue using your existing data integration tool:

- You can keep the ETL/ELT platform as it is and change the necessary storage stages with BigQuery in your ETL/ELT jobs.
- If you want to migrate the ETL/ELT platform to Google Cloud as well, you can ask your vendor whether their tool is licensed on Google Cloud, and if it is, you can install it on Compute Engine or check the Google Cloud marketplace.
- You can find some of the data integration solution providers here.

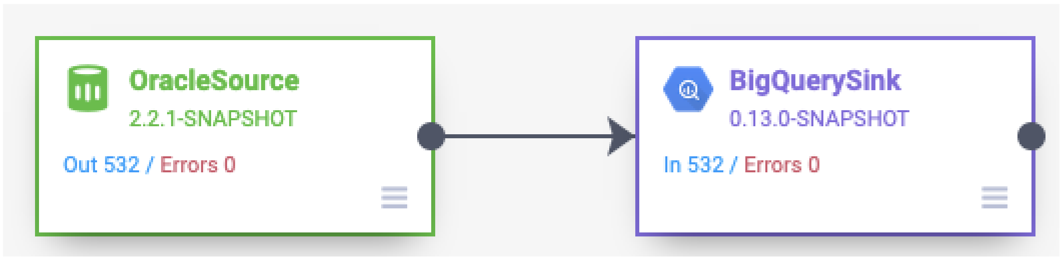
Re-architecting ETL/ELT platform

If you want to re-architect your data pipelines, we recommend that you strongly consider using Google Cloud services.

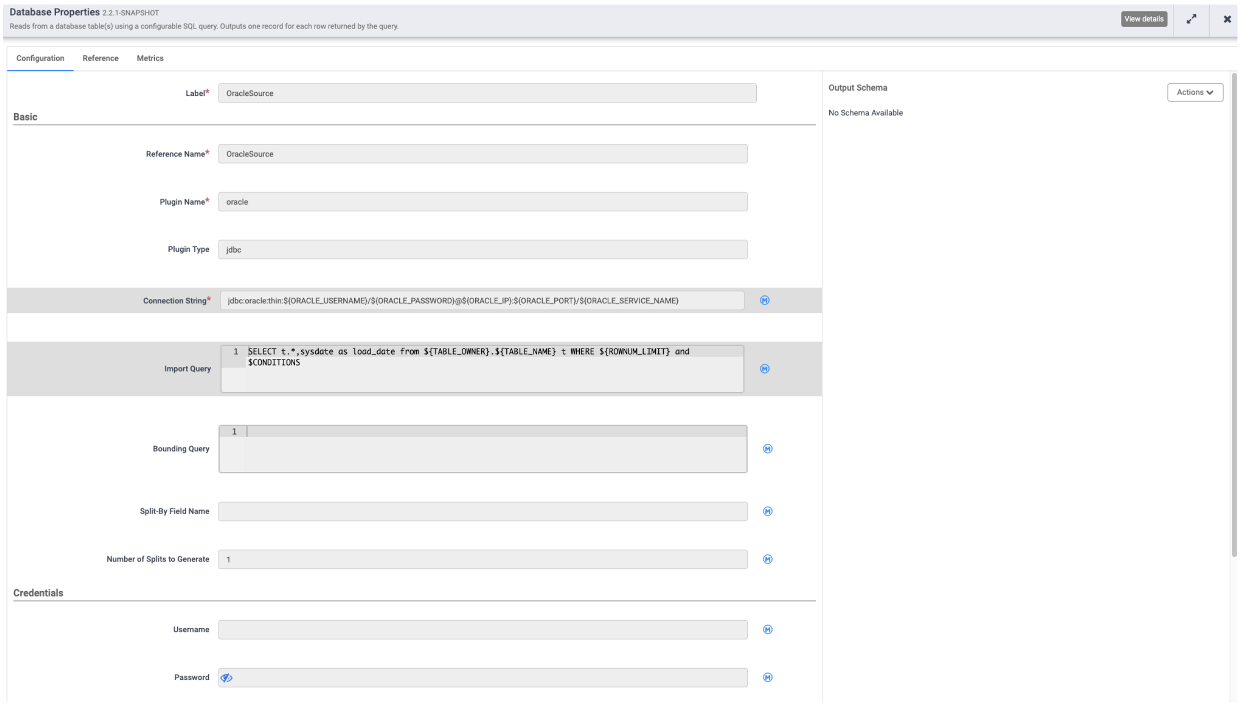
Cloud Data Fusion

Cloud Data Fusion is a managed CDAP on Google Cloud offering a visual interface with many plugins for tasks such as drag-and-drop and pipeline developments. Cloud Data Fusion can be used for capturing data from many different kinds of source systems and offers batch and streaming replication capabilities. Cloud Data Fusion or Oracle plugins can be used to capture data from an Oracle database. A BigQuery plugin can be used to load the data to BigQuery and handle schema updates.

Sample extract/load from Oracle to BigQuery using Cloud Data Fusion



Oracle source

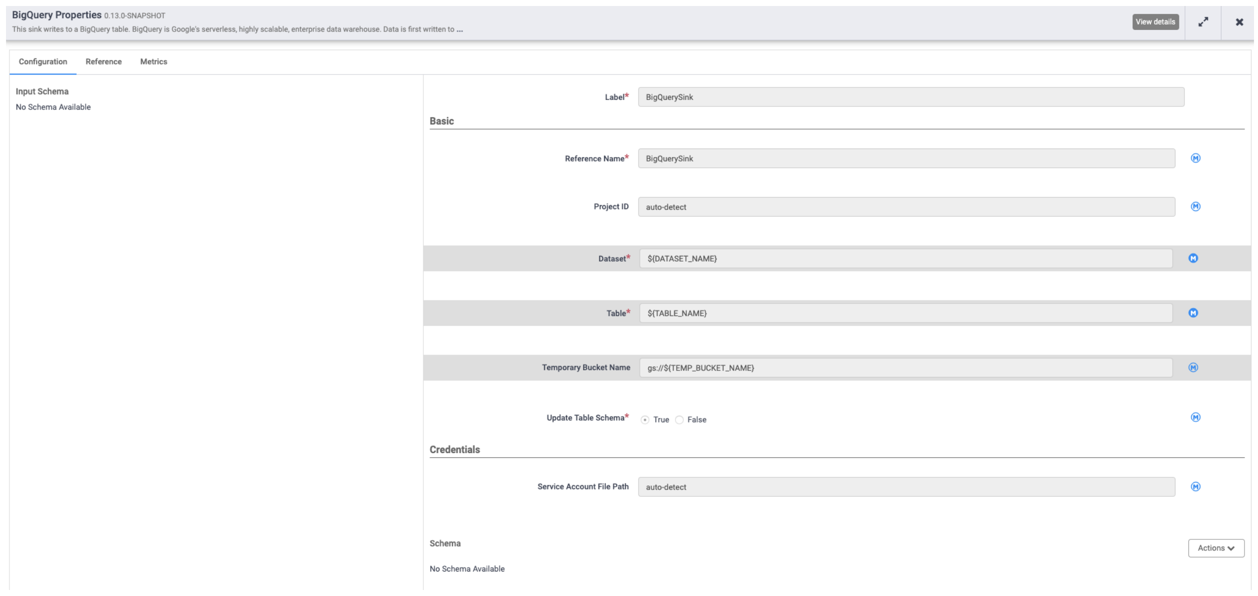


Database Properties 2.2.1-SNAPSHOT

Reads from a database table(s) using a configurable SQL query. Outputs one record for each row returned by the query.

Label*	OracleSource
Basic	
Reference Name*	OracleSource
Plugin Name*	oracle
Plugin Type	jdbc
Connection String*	jdbc:oracle:thin:${ORACLE_USERNAME}/${ORACLE_PASSWORD}@${ORACLE_IP}:${ORACLE_PORT}/${ORACLE_SERVICE_NAME}
Import Query	SELECT t.*,sysdate as load_date from ${TABLE_OWNER}.${TABLE_NAME} t WHERE ${ROWNUM_LIMIT} and $CONDITIONS
Bounding Query	
Split-By Field Name	
Number of Splits to Generate	1
Credentials	
Username	
Password	

BigQuery sink



BigQuery Properties 0.13.0-SNAPSHOT

This sink writes to a BigQuery table. BigQuery is Google's serverless, highly scalable, enterprise data warehouse. Data is first written to ...

Input Schema

No Schema Available

Label* BigQuerySink

Basic

Reference Name* BigQuerySink

Project ID auto-detect

Dataset* ${DATASET_NAME}

Table* ${TABLE_NAME}

Temporary Bucket Name gs://${TEMP_BUCKET_NAME}

Update Table Schema* [✓ True] False

Credentials

Service Account File Path auto-detect

Schema

No Schema Available

Note that no output schema is defined both on source and sink plugins, and `select * from` is used in the source plugin to replicate new columns as well.

You can use Cloud Data Fusion's Wrangle feature for data cleaning and preparing.

Dataflow

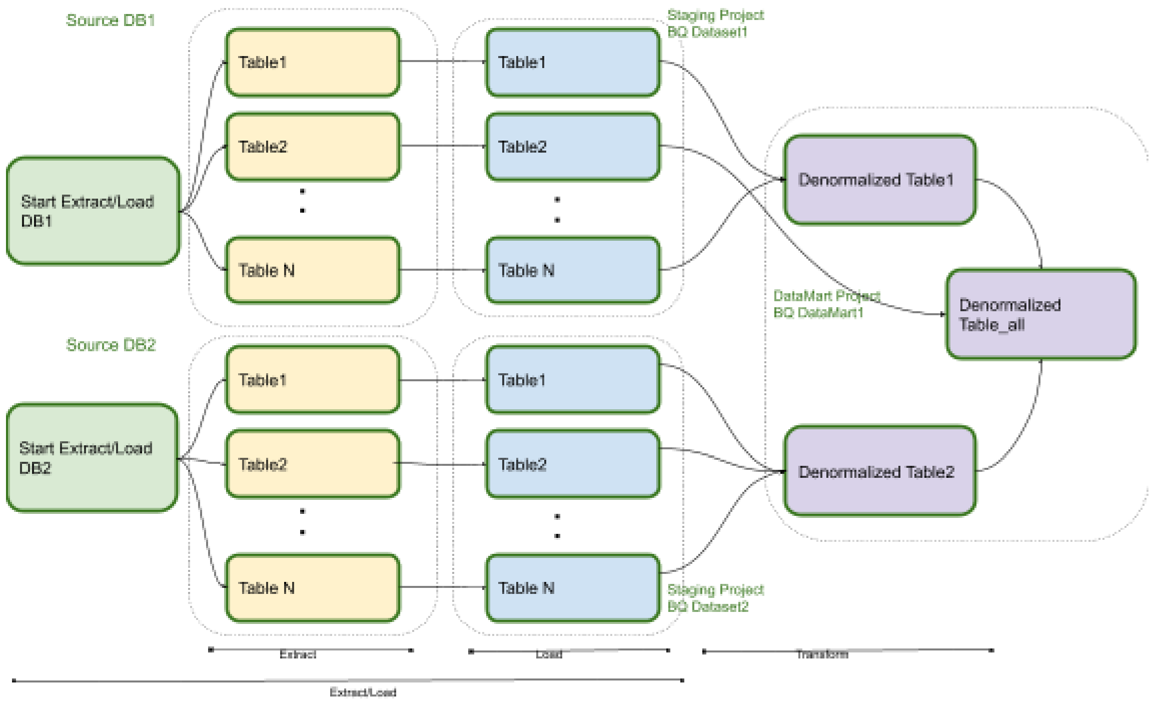
Dataflow is a serverless data processing platform that can autoscale as well as do batch and streaming data processing. Dataflow can be a good choice for Python and Java developers who want to code their data pipelines and use the same code for both streaming and batch workloads. Use the JDBC to BigQuery template to extract data from your Oracle Database or other relational databases and load it into BigQuery. An example of loading data from a relational database into a BigQuery dataset can be viewed here.

Cloud Composer

Cloud Composer is Google Cloud's fully managed workflow orchestration service built on Apache Airflow. It lets you author, schedule, and monitor pipelines that span across cloud environments and on-premises data centers. Cloud Composer provides many operators and contributions that can run multi-cloud and many technologies for many use cases including extract/loads, transformations of ELT, REST API calls, or anything that you can write in Python using the Python operator.

Cloud Composer uses directed acyclic graphs (DAGs) for scheduling and orchestrating workflows. See this link to understand general airflow concepts, and find detailed information about DAGs here. You can find sample ETL best practices with Apache Airflow here. You can replace Hive operator in that example with the BigQuery operator, and the same concepts would be applicable.

Sample DAG



Here's example code for the preceding diagram and a high-level part of a sample DAG:

```
default_args = {
  'owner': 'airflow',
  'depends_on_past': False,
  'start_date': airflow.utils.dates.days_ago(2),
  'email': ['airflow@example.com'],
  'email_on_failure': False,
  'email_on_retry': False,
  'retries': 2,
  'retry_delay': timedelta(minutes=10),
}

schedule_interval = "00 01 * * *"

dag = DAG('load_db1_db2',catchup=False, default_args=default_args,
schedule_interval=schedule_interval)

tables = {
  'DB1_TABLE1': {'database':'DB1', 'table_name':'TABLE1'},
  'DB1_TABLE2': {'database':'DB1', 'table_name':'TABLE2'},
  'DB1_TABLEN': {'database':'DB1', 'table_name':'TABLEN'},
  'DB2_TABLE1': {'database':'DB2', 'table_name':'TABLE1'},
  'DB2_TABLE2': {'database':'DB2', 'table_name':'TABLE2'},
  'DB2_TABLEN': {'database':'DB2', 'table_name':'TABLEN'},
}


start_db1_daily_incremental_load = DummyOperator(
  task_id='start_db1_daily_incremental_load',
  dag=dag
)

start_db2_daily_incremental_load = DummyOperator(
  task_id='start_db2_daily_incremental_load',
  dag=dag
)

load_denormalized_table1 = BigQueryOperator(
  task_id='load_denormalized_table1',
  use_legacy_sql=False,
  write_disposition='WRITE_TRUNCATE',
  allow_large_results=True,
  trigger_rule='all_done',
  bql='''
  #standardSQL
  select
       t1.*,tN.* except (ID)
  from `ingest-project.ingest_db1.TABLE1` as t1
  left join `ingest-project.ingest_db1.TABLEN` as tN
  on t1.ID = tN.ID
  ''',    destination_dataset_table='datamart-project.dm1.dt1',
  dag=dag)

load_denormalized_table2 = BigQueryOperator(
  task_id='load_denormalized_table2',
  use_legacy_sql=False,
  write_disposition='WRITE_TRUNCATE',
  allow_large_results=True,
  trigger_rule='all_done',
  bql='''
  #standardSQL
  select
       t1.*,t2.* except (ID),tN.* except (ID)
  from `ingest-project.ingest_db1.TABLE1` as t1
  left join `ingest-project.ingest_db2.TABLE2` as t2
  on t1.ID = t2.ID
  left join `ingest-project.ingest_db2.TABLEN` as tN
  on t2.ID = tN.ID
  ''',    destination_dataset_table='datamart-project.dm1.dt2',
  dag=dag)

load_denormalized_table_all = BigQueryOperator(
  task_id='load_denormalized_table_all',
  use_legacy_sql=False,
  write_disposition='WRITE_TRUNCATE',
  allow_large_results=True,
  trigger_rule='all_done',
  bql='''
  #standardSQL
  select
       t1.*,t2.* except (ID),t3.* except (ID)
  from `datamart-project.dm1.dt1` as t1
  left join `ingest-project.ingest_db1.TABLE2` as t2
  on t1.ID = t2.ID
  left join `datamart-project.dm1.dt2` as t3
  on t2.ID = t3.ID
  ''',    destination_dataset_table='datamart-project.dm1.dt_all',
  dag=dag)

def start_pipeline(database,table,...):
  #start initial or incremental load job here
  #you can write your custom operator to integrate ingestion tool
  #or you can use operators available in composer instead


for table,table_attr in tables.items():
  tbl=table_attr['table_name']
  db=table_attr['database'])
  load_start = PythonOperator(
    task_id='start_load_{db}_{tbl}'.format(tbl=tbl,db=db),
    python_callable=start_pipeline,
    op_kwargs={'database': db, 'table':tbl},
    dag=dag
  )

  load_monitor =  HttpSensor(
    task_id='load_monitor_{db}_{tbl}'.format(tbl=tbl,db=db),
    http_conn_id='ingestion-tool',
    endpoint='restapi-endpoint/',
    request_params={},
    response_check=lambda response: """{"status":"STOPPED"}""" in
response.text,
    poke_interval=1,
    dag=dag,
  )
  load_start.set_downstream(load_monitor)

  if table_attr['database']=='db1':
    load_start.set_upstream(start_db1_daily_incremental_load)
  else:
    load_start.set_upstream(start_db2_daily_incremental_load)

  if table_attr['database']=='db1':
    load_monitor.set_downstream(load_denormalized_table1)
  else:
    load_monitor.set_downstream(load_denormalized_table2)

load_denormalized_table1.set_downstream(load_denormalized_table_all)
load_denormalized_table2.set_downstream(load_denormalized_table_all)
```

Note that the preceding code is provided for demonstration purposes and cannot be used as it is.

Dataprep by Trifacta

Dataprep by Trifacta is an intelligent data service for visually exploring, cleaning, and preparing structured and unstructured data for analysis, reporting, and machine learning. You export the source data into JSON or CSV files, transform the data using Dataprep, and load the data using Dataflow. For an example, see here.

Dataproc

Dataproc is Google's managed Hadoop service. You can use Sqoop to export data from Oracle and many relational databases into Cloud Storage as Avro files, and then you can load Avro files into BigQuery using the bq command-line tool. It is very common to install ETL tools like CDAP on Hadoop that use JDBC to extract data and Apache Spark or MapReduce for transformations of the data.

Partner tools for data migration

There are several vendors in the extraction, transformation, and load (ETL) space. ETL market leaders such as Informatica, Talend, Matillion, Alooma, Infoworks, Stitch, Fivetran, and Striim have deeply integrated with both BigQuery and Oracle and can help extract, transform, and load data as well as manage processing workflows.

ETL tools have been around for many years. Some organizations might find it convenient to leverage an existing investment in trusted ETL scripts. Some of our key partner solutions are included on Google Cloud's BigQuery partner website. Knowing when to choose partner tools over Google Cloud's built-in utilities depends on your current infrastructure and your IT team's comfort with developing data pipelines in Java or Python code. Use the following flow chart as a general guideline about when to choose a partner ETL tool.

Note that you can use one or more tools from the Ingestion Components area or any other tool that supports BigQuery, on-premises, or another cloud environment. You can also use some Cloud Composer operators for directly ingesting into BigQuery.

Projects are separated to provide least-privileged access to resources, reduce blast radius, and enable different teams to work autonomously without affecting each other. This is a sample; the high-level architecture, resource hierarchy, project setup, and resources might vary depending on the constraints.

Business intelligence (BI) tool migration

BigQuery supports a flexible suite of business intelligence (BI) solutions for reporting and analysis that you can leverage. For more information about BI tool migration and BigQuery integration, see Migrating data warehouses to BigQuery: Reporting and analysis.

Query (SQL) translation

BigQuery standard SQL supports compliance with the SQL 2011 standard and has extensions that support querying nested and repeated data. All ANSI-compliant SQL functions and operators can be used with minimal modifications. For a detailed comparison between Oracle and BigQuery SQL syntax and functions, see the Oracle to BigQuery SQL translation reference.

Migrating Oracle Database options

This section presents architectural recommendations and references for converting applications that use Oracle Data Mining, R, and Spatial and Graph functionalities.

Oracle Advanced Analytics option

Oracle offers advanced analytics options for data mining, fundamental machine learning (ML) algorithms, and R usage. The Advanced Analytics option requires licensing. You can choose from a comprehensive list of Google AI/ML products depending on your needs from development to production at scale.

Oracle R Enterprise

Oracle R Enterprise (ORE), a component of the Oracle Advanced Analytics option, makes the open source R statistical programming language integrate with Oracle Database. In standard ORE deployments, R is installed on an Oracle database server.

For very large scales of data or approaches to warehousing, integrating R with BigQuery is an ideal choice. You can use the open source bigrquery R library to integrate R with BigQuery. Google has partnered with RStudio to make the field's cutting-edge tools available to users. RStudio can be used to access terabytes of data in BigQuery, fit models in TensorFlow, and run machine learning models at scale with AI Platform. In Google Cloud, R can be installed on Compute Engine at scale.

Oracle Data Mining

Oracle Data Mining (ODM), a component of the Oracle Advanced Analytics option, lets developers build machine learning models using Oracle PL/SQL Developer on Oracle Database.

BigQuery ML enables developers to run linear regression, binary logistic regression, multiclass logistic regression, k-means clustering, and TensorFlow model imports. For more information, see Introduction to BigQuery ML.

Converting ODM jobs might require rewriting the code. You can choose from comprehensive Google AI product offerings such as BigQuery ML, AI APIs (Speech-to-Text, Text-to-Speech, Dialogflow, Cloud Translation, Cloud Natural Language API, Vision API, Cloud Inference API, and more), AutoML, AutoML Tables, or AI Platform.

Google AI Platform Notebooks can be used as a development environment for data scientists, and Google AI Platform Training can be used to run training and scoring workloads at scale.

Spatial and Graph option

Oracle offers the Spatial and Graph option for querying geometry and graphs and requires licensing for this option. You can use the geometry functions in BigQuery without additional costs or licenses and use other graph databases in Google Cloud.

Spatial

BigQuery offers GIS functions and data types. For more information, see Working with BigQuery GIS data. Oracle Spatial data types and functions can be converted to geography functions in BigQuery standard SQL. Geography functions don't add cost on top of standard BigQuery pricing.

Graph

JanusGraph is an open source graph database solution that can use Cloud Bigtable as a storage backend; here is a guide for deploying JanusGraph with Cloud Bigtable on Google Cloud.

Neo4j is another graph database solution delivered as a Google Cloud service that runs on Google Kubernetes Engine (GKE).

Oracle Application Express

Oracle Application Express (APEX) applications are unique to Oracle Database and need to be rewritten. Reporting and data visualization functionalities can be developed using Data Studio or BI engine, whereas application-level functionalities such as creating and editing rows can be developed without coding on AppSheet using Cloud SQL.

Release notes

See the release notes, and stay up to date on the latest features and developments in BigQuery.