

Data Vault 2.0 on Google Cloud BigQuery

Valentino Miazzo, Christine Schulze, Alfred Geers, Firat Tekiner

1. What is the Data Vault model?

Page 4

2. Why BigQuery for Data Vault?

Page 6

3. Data Vault implementation

Page 7

4. Conclusions to set up a Data Vault model on BigQuery

Page 18



Abstract

Data Vault is a data warehouse methodology developed to solve the challenges faced when maintaining and evolving large data warehouse projects. **In brief, it aims to combine the centralized data repository (Inmon) with iterative bottom-up data mart modeling (Kimball).** This paper provides an overview of the Data Vault concept and the business benefits of leveraging it on the cloud-based enterprise database BigQuery. Also, it serves as a comprehensive guide on how to set up a Data Vault model on BigQuery with detailed “how-tos” and possible pitfalls to support implementing a solid and future-proofed Data Vault model in BigQuery.

Introduction

Data warehouses have been the custodians of enterprises' most valuable business data for the past few decades. As enterprises are increasingly becoming data-driven, data warehouses play a critical role in enterprises' digital transformation journey. Data warehouses often form the foundation of enterprise analytics strategy. **However, while the world is generating more data than ever before, enterprises are unable to analyze all the relevant data.**

Today's enterprises are unable to leverage the power of big data to improve business processes¹.

Traditional data warehouses were not designed to handle the explosive data growth. To deal with the massive amounts of data and answer increasingly complex questions from the users, Hadoop / Spark were introduced in the last decade. It was clear that data analysis tools outside of the data warehouses were missing something. However, traditional data warehouses are either too costly or not elastic enough to respond to the changes.

Cloud computing has changed the way that we approach data. Traditionally, organizations have had to manage large amounts of infrastructure to extract value from data, starting with data warehouses and leading to the rise of Hadoop-based data lakes. However, both approaches have their challenges, and we are in a transformative technical era in cloud computing where we can leverage the best of both worlds.

Google has gone through this transformation, too. In fact, Google's data processing environment is built with digital transformation in mind. BigQuery acts as a massive data warehouse, hosting and processing exabytes of data.

Google BigQuery is fully managed and serverless. Storage and compute are decoupled and can scale independently on-demand. This offers enormous flexibility and cost control for your business as you don't need to keep your expensive compute resources up and running all the time. This is very different from the traditional node-based cloud data warehouse solutions or on-premise MPP systems.

Data Vault on the other hand is a data warehouse methodology developed to solve the challenges faced when maintaining and evolving large data warehouse projects. DataVault is an agile way of designing and building a Data Warehouses. **In brief, it aims to combine the centralized data repository (Inmon) with iterative bottom-up data mart modeling (Kimball)².** Data Vault has been gaining popularity in recent years. This paper provides an overview of the Data Vault concept and the business benefits of leveraging it on the cloud-based enterprise database BigQuery. Also, it serves as a comprehensive guide on how to set up a Data Vault model on BigQuery with detailed "how-tos" and possible pitfalls to support implementing a solid and future-proofed Data Vault model in BigQuery.

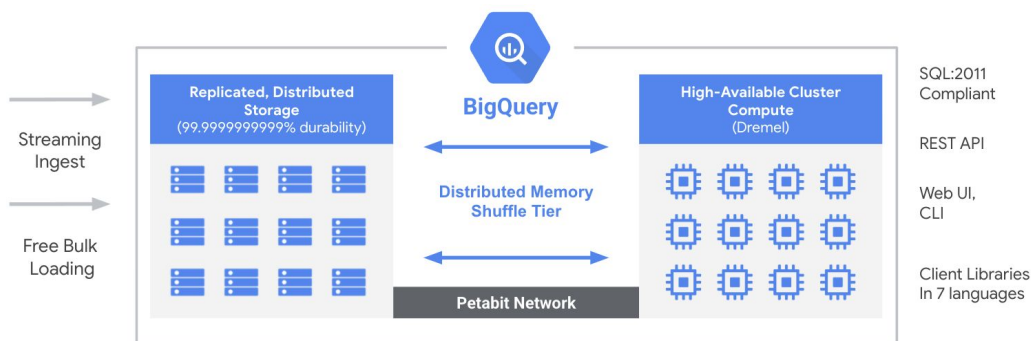


Figure 1: BigQuery architecture

¹ <https://www.accenture.com/us-en/insights/technology/closing-data-value-gap>

² <https://www.computerweekly.com/tip/Inmon-or-Kimball-Which-approach-is-suitable-for-your-data-warehouse>

1. What is the Data Vault model?

In a classic data warehouse environment, managing changes is often a painful task. Adjustments in the data model are complex and there are several development teams involved which leads to time-consuming implementation and testing. When using the Data Vault model, all this effort is limited to when new sources, entities, and KPIs are added because the model itself avoids refactorization. Data Vault is a hybrid evolution of previous approaches and has been designed for change management and to simplify the integration of diverse sources. **Data Vault is optimized to avoid load-time dependencies between tables. This permits you to load tables in parallel when taking advantage of massively parallel, autoscaling serverless systems like BigQuery.**

A Data Vault model consists of 3 concepts: Hubs, Links, and Satellites. A Hub table collects all the business keys present in a source entity. A Link table represents (in an N to N relationship) a relationship and uses the business keys to connect two Hubs. Satellites store all data that describes a row in a Hub or a Link. Satellites capture not only the existence but also the evolution over time of business objects or relationships. This concept separates structural information from descriptive data. **The number of data sources can continuously grow by adding new Hubs, and from there new Links and Satellites, without ever changing the existing ones.**

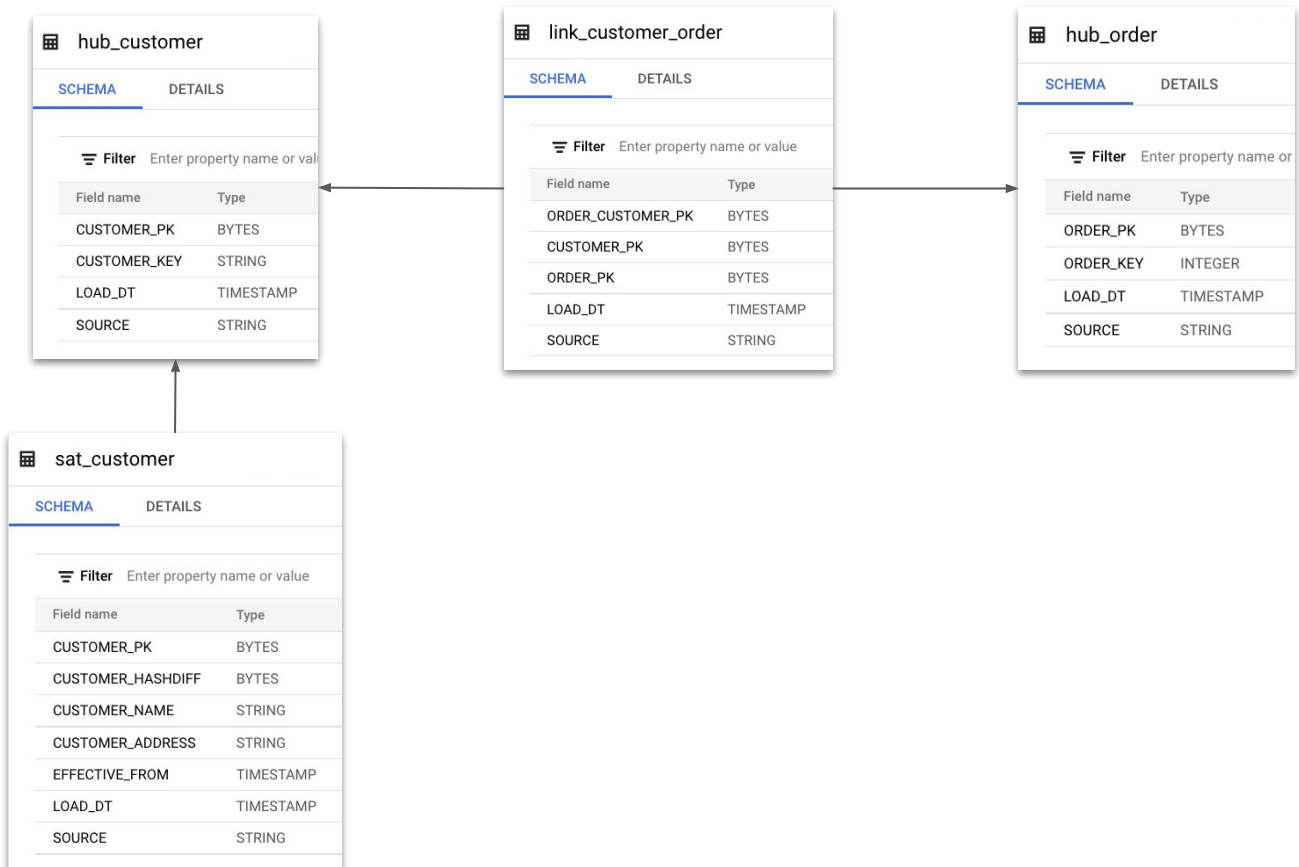


Figure 2: Hubs, Links, and Satellites

A Data Vault architecture may consist of the following components:

- The **Staging Layer** does not apply any changes to the data. It provides a temporary area to support the process of moving data from various sources into the data warehouse.
- The **Raw Vault** stores persistently the data in a model built around the identification of the business keys in the data sources. The model supports new data sources or schema changes in the existing sources without requiring the modifications of the existing Raw Vault tables. This helps to satisfy strict auditability requirements. To minimize the need for refactorings due to ever-changing business rules, data is stored in the Satellites as-is. The only decisions taken at this stage are those related to this copy (e.g. to choose data types) and are called Hard Business Rules.
- The **Business Vault** is built from the Raw Vault retaining the Hub, Link, Satellite structure and applying all the transformations needed to conform the data to the business rules defined by the enterprise (derived and curated data). Since these rules change as often as the business, they are called Soft Business Rules. The Business Vault has no strict persistence requirements. Depending on the requirements, it can be made of views and materialized/rebuilt as needed. This separation of concerns between the two layers is key in the Data Vault design: the Raw Vault is about scalable ingestion of new data sources while the Business Vault is about evolving business requirements.
- The presentation layer consists of one or more **Information Marts** intended to be requirements-driven, scope-specific subsets of the data warehouse data. It is needed because data in the Business Vault is still in the Hub, Link, Satellite shape and has to be shaped as expected by the final business user (e.g. star schema, feature store for data mining, etc). It can be made of views or rebuilt as needed.

All these layers can be stored and maintained in separated BigQuery datasets. Data movement and management are just transformations from one table to another using the full scalability and performance of BigQuery without the need for cluster or node management etc. Data Vault defines other optional elements that are not described in this paper.

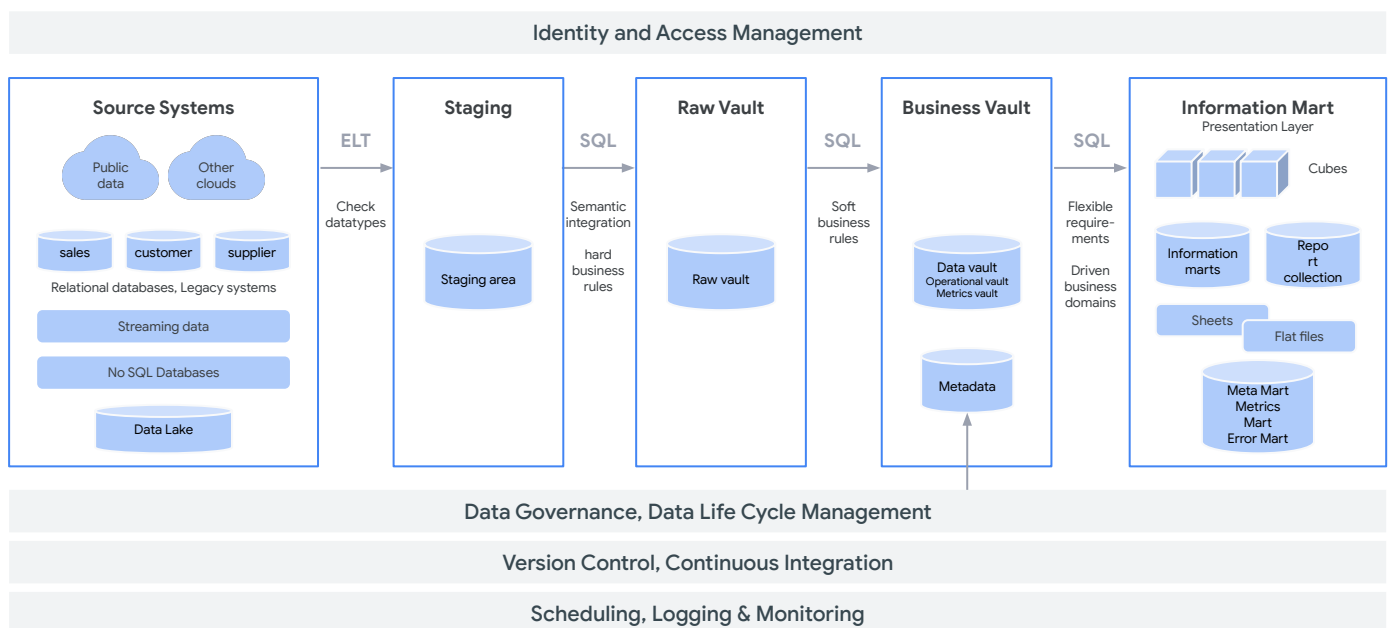


Figure 3: Data Vault staging architecture

2. Why BigQuery for Data Vault?

BigQuery is Google's serverless, cloud-native, analytical database. As a relational columnar database, BigQuery is an ideal system for implementing the Data Vault concept.

1. Data Vault follows the [open-closed principle](#) and therefore is always adding new tables, meaning the vault grows organically. It also supports parallel loading. This matches with [BigQuery's](#) serverless scaling out architecture and the massive parallelization of compute power. The decoupling of storage and compute layers permits you to always obtain the optimal price point and avoid the overprovisioning that is typical of architectures where compute is bound to storage.
2. Data Vault supports a volatile data environment and embraces change. For example, during refactorings, additional temporary storage is required or bursts of computation are performed that deviate from the typical usage. Hence, BigQuery is the right choice with its serverless architecture and its capacity in scaling up and down without any disruptions.
3. BigQuery [in-memory query execution](#), [columnar storage](#), and [smart caching](#) accelerate data processing. Speed is critical to give data analysts fast access to the presentation layer, minimize ingestion delays, and complete refactorings in a short time.
4. Data Vault supports the integration with big data platforms via hashed keys. BigQuery not only has built-in [hashing functions](#) but it seamlessly complements this with the ability to read big data in place via [external data sources](#).
5. BigQuery has built-in [machine learning](#) capabilities that go beyond the standard business intelligence allowed by Data Vault to address tasks like forecasting, classification, and anomaly detection.
6. Data Vault is architected to support a high level of automation, keeping the focus on managing insight, not ingestion. This is highly supported by the cloud-native and serverless architecture of BigQuery, which minimizes admin functionality. BigQuery allows you to start small and grow without ever refactoring the infrastructure.

The following diagram summarizes the advantages in performance when using BigQuery for Data Vault architecture.

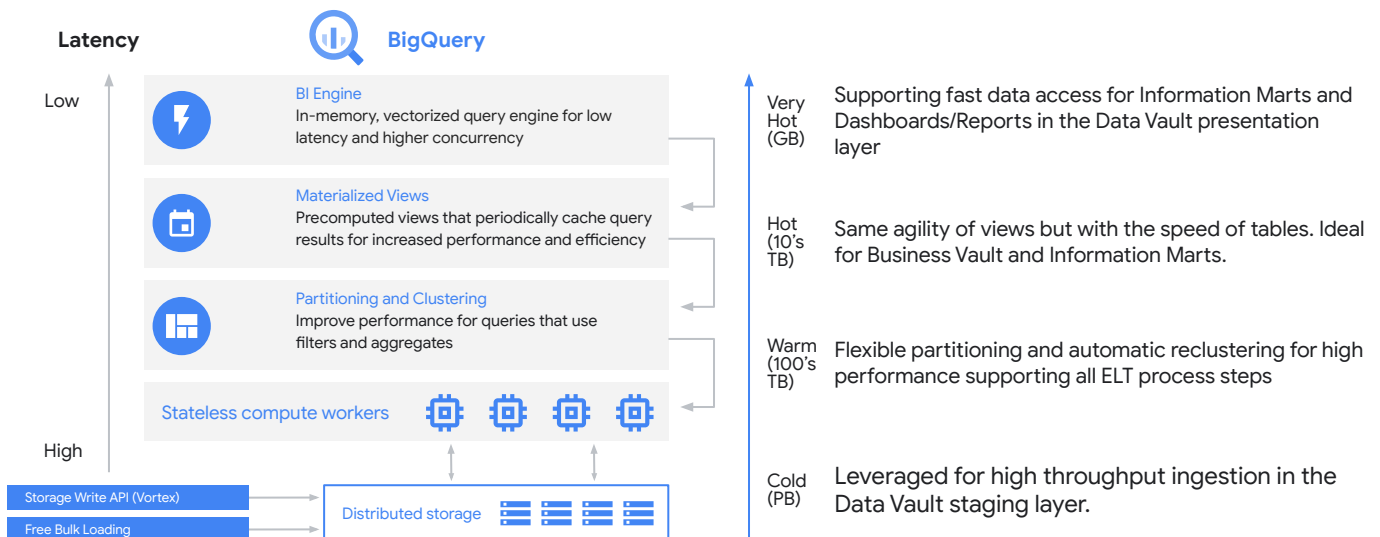


Figure 4: BigQuery for Data Vault

3. Data Vault implementation

Let's look into how Data Vault concepts apply to BigQuery. After some initial general architectural discussion, we will show simple examples of how to implement the various elements of a Data Vault model like Satellites, Hubs, and Links and how BigQuery supports this. Further, we will discuss some specific optimization techniques for the Business Vault and the Information Marts.

3.1. General consideration

3.1.1 Extract load transform

ELT stands for extract load transform and describes an alternative approach to data integration and processing than ETL which stands for extract, transform, load. ELT is the best approach with BigQuery because ELT processes data inside BigQuery gaining the advantage of its extreme parallelism and bisectional bandwidth. On the other hand, with the ETL approach, data is read from BigQuery, processed elsewhere and then written back. The net result is that ELT is invariably cheaper and faster. ETL was mostly used in the past to offload constrained traditional data warehouses.

3.1.2 Think columnar

Data Vault literature sometimes assumes a row-oriented database and proposes patterns that are useless or even counterproductive with a column-oriented database. It is recommended to not use patterns blindly and instead reinterpret them in the light of the database you are using. For example, on BigQuery we don't have to split Satellites, or other tables, if they are too wide. Also see chapter 3.3.3 for a discussion about `LOAD_DT` column partitioning.

3.1.3 Project structure

In the Google Cloud Platform, resources are administered and organized in a [hierarchy](#) where projects can be viewed as a leaf of a tree structure (organization, folders, projects). BigQuery [further organizes](#) data in datasets and tables.

There are a number of ways this structure can be set up for organizations. The setup below will be used as a reference within this article.

- **Project: `dwh-stor-staging`**³. This project is used to scope resources dedicated to the Staging Layer. The Staging Layer is connected to multiple systems. It is useful to manage it differently from the other parts of the DWH. For example, we may want some specific [network security settings](#) for this project.
 - **Dataset: `stage`**. Is a dataset for the Staging Layer in BigQuery.
- **Project: `dwh-stor-vault`**. This project contains the tables that compose the Data Vault. It is not directly accessed by users and other systems and is managed separately.
 - **Dataset: `raw`**. Dataset for the Raw Vault.
 - **Dataset: `business`**. Dataset for the Business Vault.
- **Project: `dwh-comp-vault`**. Is used for workload management which will be discussed in detail in the next section.
- **Project: `dwh-mart-1..N`**. Information Marts are more volatile than the rest of the DWH. Over time, they are created and discarded to reflect the evolution of the business. They can also be tied to specific groups of users. Assigning a dedicated project to each data mart also simplifies authorization, lifecycle, and cost management. There can be multiple Information Mart projects reflecting different business requirements.
 - **Dataset: `mart1`**. Dataset for an individual Mart

³ Names are indicative.

Google Cloud

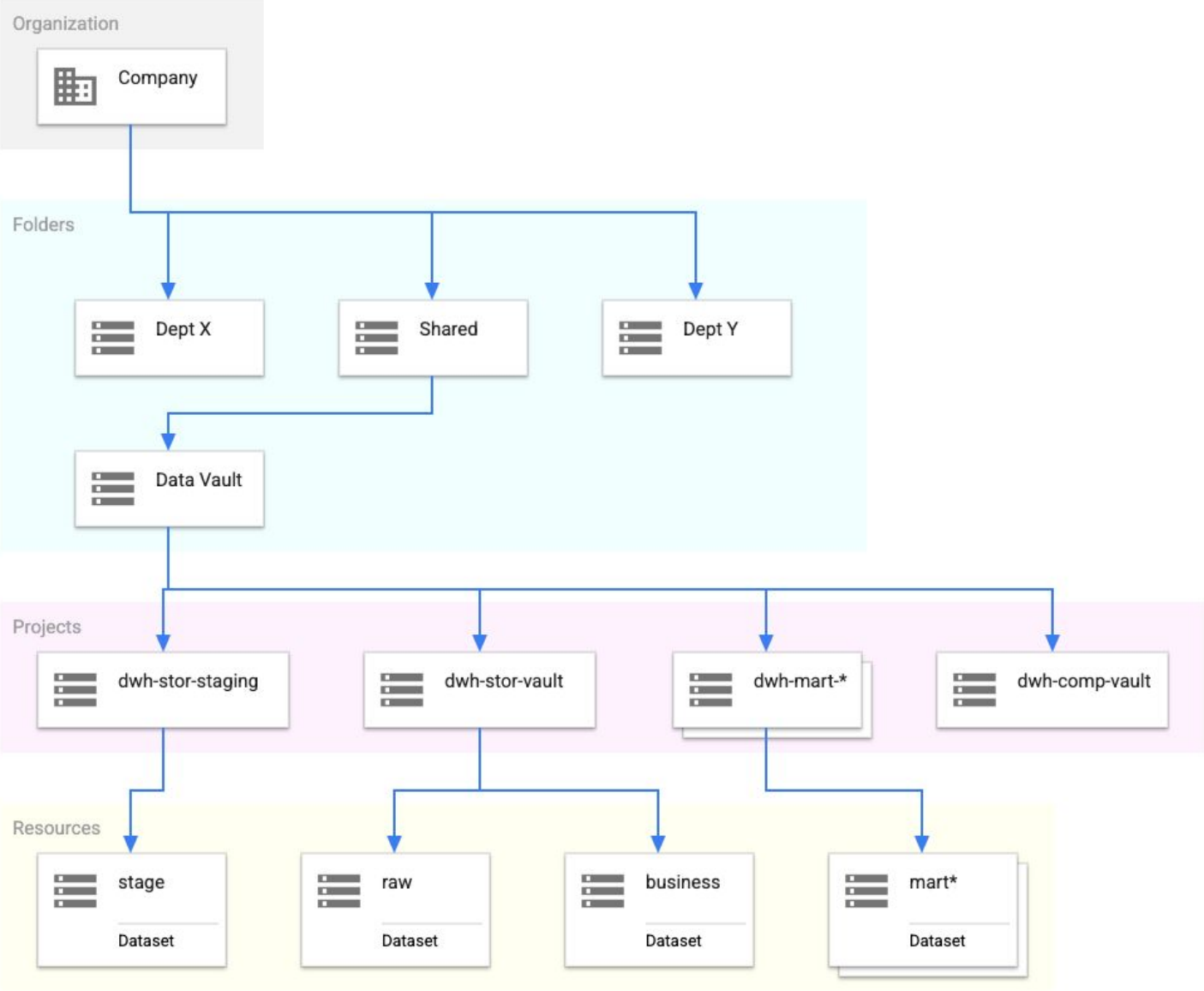


Figure 5: A possible resource hierarchy for Data Vault 2

3.1.4 Workload management

In BigQuery the compute resources available to a query are determined by the Origin Project⁴ used by the query. In the web UI, the Origin Project is the one selected by the project selector at the top left.

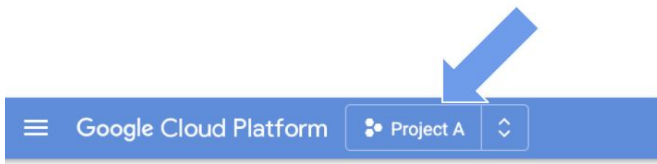


Figure 6: Project selector in BigQuery

Workload management refers to the assignment of different amounts of compute resources to different workloads. It is entirely optional but common for cost control. [BigQuery Reservations](#) are the tool that allows us to assign compute resources to projects⁵. Even if this is not a strictly required setup, we think it is important to embed the concept of Origin Project from the beginning as a parameter in automation. Referring to the project structure presented above, we follow the rule of using dwh-comp-vault as the Origin Project for our Data Vault pipelines.

3.2. Staging Layer

3.2.1 Ingestion

The Staging Layer is where the data is copied from other systems. Data is copied in as-is and from there ingested into the Raw Vault and then deleted. Data may stay here for some days if there are technical issues that are being solved by the DWH team but, in any case, this data is temporary. There are multiple ingestion options such as [batch load](#) (which is free), [rows streaming](#), [CDC](#), [Data Transfer Services](#), [Federation](#), or many commercial ETL Tools.

3.2.2 Personal identifiable information

It is important that Personal Identifiable Information (PII) enters the Data Vault in a controlled way. Since the Raw Vault contains the unmodifiable history of all the data ingested, unwanted PII should be discovered and filtered before it is ingested in the Raw Vault or, alternatively, required PII data may need to be masked. [Google Data Loss Prevention](#) can [inspect BigQuery tables](#) and automate PII discovery via [data profiles](#). It can also de-identify, re-identify and produce re-identification risk metrics.

3.2.3 Schema on read

Though organizations load data from semi-structured sources that may contain hundreds of fields, they tend to be interested in only a few of the fields and ignore the others. BigQuery simplifies this through Google Cloud Storage federation, [schema autodetect](#), [JSON datatype](#) and [JSON functions](#). With very cheap cloud storage, there is no need to pre-filter attributes. We recommend loading everything at least into the Raw Vault. Promotion to the Business Vault can then be done when it will make business sense.

3.3. Raw Vault

With BigQuery, as an ANSI-compliant relational database, the Raw Vault implementation is very similar to other RDBMS implementations.

3.3.1 Hubs and Links

Hubs and Links do not contain any business data and are never versioned, so the population logic is quite straightforward. For example, the following snippet creates a simple Hub table in BigQuery.

⁴ The official documentation calls it the “Billing Project”. The authors think that “Origin Project” is clearer in this context.

⁵ More details [here](#).

```
CREATE OR REPLACE TABLE
dwh-stor-vault.raw.hub_customer
(
  CUSTOMER_PK BYTES
  , CUSTOMER_KEY STRING
  , LOAD_DT TIMESTAMP
  , SOURCE STRING
)
```

A fully qualified name in BigQuery contains the Project (e.g. *dwh-stor-vault*), the dataset (e.g. *raw*), and the Table name (e.g. *hub_customer*). Datasets are logical grouping structures within BigQuery. We recommend creating a separate dataset for Staging Layer, Raw Vault, Business Vault, etc.

Data Vault recommends creating surrogate primary keys using MD5 or SHA. BigQuery has [built-in support](#) to create these hash keys. The following snippet shows an example load statement using the built-in MD5 function. Alternatively, SHA256 or SHA512 could be used for high collision scenarios.

The LOAD_DT column marks the first time the key is seen by the DWH.

```
INSERT INTO
dwh-stor-vault.raw.hub_customer
SELECT DISTINCT
MD5(IFNULL(NULLIF(UPPER(TRIM(CAST(s.CUSTOMER_KEY
AS STRING))), ''), '^A')) AS CUSTOMER_PK
, s.CUSTOMER_KEY
, s.LOAD_DT
, s.SOURCE
FROM
dwh-stor-staging.stage.customer AS s
LEFT JOIN
dwh-stor-vault.raw.hub_customer AS r
ON
r.CUSTOMER_KEY = s.CUSTOMER_KEY
WHERE
r.CUSTOMER_KEY IS NULL
AND s.LOAD_DT = TIMESTAMP('2021-10-10
00:00:00+00')
```

In the example above, we insert in the customer Hub of the Raw Vault the new customers coming from the Staging Layer using a join to skip keys already in the Hub. Note that we are making some assumptions here. These are Hard Business Rules since they are applied at the ingestion into the Raw Vault and are therefore irreversible. First, we assume keys are strings and can contain leading or trailing spaces that are irrelevant. Second, we assume keys are case insensitive. Third, we assume that keys empty, or composed only of spaces, should be handled like NULL keys. Fourth, we do not allow for NULL surrogate primary keys. We join on the key and not on the surrogate primary key to account for (unlikely) hash collisions.

Of course, this logic must be adapted to the specific business situation. In the real world, requirements are sometimes more complex. BigQuery functionalities like [user-defined functions](#) (UDF) can be used to encapsulate such logic.

3.3.2 Satellites

Various techniques can be used with BigQuery to optimize the load and the access of Satellites. Initially, lets represent the basic setup and then explore optimizations. Similar to Hubs, the CREATE statement of a Satellite is quite straightforward. It can be done simply by providing the surrogate primary key (PK), the customer context date, and a HASHDIFF column to simplify the delta detection for incremental load as suggested by Data Vault.

```
CREATE OR REPLACE TABLE
dwh-stor-vault.raw.sat_customer
(
  CUSTOMER_PK BYTES
  , CUSTOMER_HASHDIFF BYTES
  , CUSTOMER_NAME STRING
  , CUSTOMER_ADDRESS STRING
  , EFFECTIVE_FROM TIMESTAMP
  , LOAD_DT TIMESTAMP
  , SOURCE STRING
)
```

And the following INSERT statement can be used to populate the table.

```
INSERT INTO dwh-stor-vault.raw.sat_customer
WITH
newcust AS (
  SELECT DISTINCT
    CUSTOMER_NAME,
    CUSTOMER_ADDRESS,
    EFFECTIVE_FROM,
    LOAD_DT,
    SOURCE,
    MD5(IFNULL(NULLIF(UPPER(TRIM(CAST(CUSTOMER_KEY AS STRING))), ''), '^^')) AS CUSTOMER_PK,
    MD5(CONCAT(
      IFNULL(NULLIF(UPPER(TRIM(CAST(CUSTOMER_ADDRESS AS STRING))), ''), '^^'), '||',
      IFNULL(NULLIF(UPPER(TRIM(CAST(CUSTOMER_NAME AS STRING))), ''), '^^'), '||',
      IFNULL(NULLIF(UPPER(TRIM(CAST(CUSTOMER_KEY AS STRING))), ''), '^^') )) AS CUSTOMER_HASHDIFF
  FROM dwh-stor-staging.stage.customer
  WHERE LOAD_DT = TIMESTAMP('2021-10-10 00:00:00+00')
),
currcust AS (
  SELECT
    CUSTOMER_PK,
    CUSTOMER_HASHDIFF,
    RANK() OVER (PARTITION BY CUSTOMER_PK ORDER BY EFFECTIVE_FROM DESC) AS RNK
  FROM dwh-stor-vault.raw.sat_customer
  WHERE TRUE QUALIFY RNK = 1
)
SELECT
  newcust.CUSTOMER_PK,
  newcust.CUSTOMER_HASHDIFF,
  newcust.CUSTOMER_NAME,
  newcust.CUSTOMER_ADDRESS,
  newcust.EFFECTIVE_FROM,
  newcust.LOAD_DT,
  newcust.SOURCE
FROM newcust
LEFT JOIN currcust
USING (CUSTOMER_PK, CUSTOMER_HASHDIFF)
WHERE
  currcust.CUSTOMER_HASHDIFF IS NULL
```

Let's walk over some specifics of the above statement.

HASHDIFF: It is created by concatenating all columns within the `sat_customer` table and executing the MD5 function within BigQuery. By including the KEY in the MD5 calculation, we ensure that rows with the same data but different keys have different HASHDIFF. In this example, by using UPPER, we implicitly impose the Hard Business Rule that ADDRESS, NAME, KEY are case insensitive. You may want to postpone this decision and implement these as Soft Business Rules when creating the Satellites in the Business Vault.

Aliases: At the time of writing, BigQuery does not allow you to reference aliases defined in the select clause directly from the where clause. It is therefore common with BigQuery to use the WITH statement structure to define the alias and subsets first and then use them in select or insert statements. However, it should be noted that subqueries, inline views, [multi-statement transactions](#) and [temporary tables](#) are also fully supported with BigQuery.

EFFECTIVE_FROM: The EFFECTIVE_FROM column is optionally provided by the source system and marks the instant the row was updated in the source system. The clock of the source system can be out of sync with the DWH. For this reason, it should be used only to sort satellites rows in time and when the load cycles are not aligned with the business dates. If not available, you can use the LOAD_DT value.

Delta Detection: Delta loads compare the new context data with the existing context data to detect changed information. As recommended by Data Vault, this can be done using the comparison of the HASHDIFF values. The information in Satellites is in general historicized in a Data Vault, so such comparison has to be done only for the most recent record. In the above example, we used a SQL window function to determine, per PK value, the record with the latest EFFECTIVE_FROM value. Also, we used the RANK function as an example but it should be noted

that alternative implementations using MAX(EFFECTIVE_FROM) or ROW_NUMBER would perform similarly.

On some analytical systems, SQL window functions can suffer from poor parallelization in presence of data skew. BigQuery dynamically changes the execution plan and repartitions the data for optimal execution time when such cases occur and it finds skew. Then, using window functions with BigQuery tends to lead to good performance. For comparison, we also tested alternative implementations, like using a correlated subquery to find the latest records, and found that the window functions approach is in general significantly faster.

3.3.3 Optimizations

One of the main values of using a Data Vault modeling approach is the efficient and agile load of data. This comes with the burden of having a complex data model and access overhead due to the extensive need for joins or complex queries to take the historicization into account. Some of this will be discussed later in the context of the Business Vault. Here we want to discuss some BigQuery optimization techniques addressing the physical data model.

Struct Data Type: Besides typical ANSI data types, BigQuery supports more complex data types like structs or arrays. Often, STRUCT or JSON attributes are used to support semi-structured types of information or as a tuning element to avoid join. For example, in Data Vault we could model every Satellite as a STRUCT attribute within the Hub tables as in the example below.

```
CREATE OR REPLACE TABLE
dwh-stor-vault.raw.hub_customer
(
  CUSTOMER_PK BYTES,
  ,CUSTOMER_KEY STRING
  , SAT_CUSTOMER STRUCT<
    CUSTOMER_NAME STRING
    ,CUSTOMER_ADDRESS STRING
    ,EFFECTIVE_FROM TIMESTAMP
    ,CUSTOMER_HASHDIFF BYTES
  >
  ,LOAD_DT TIMESTAMP,
  ,SOURCE STRING
)
```

Based on our analysis of the STRUCT data type for Data Vault, we do not recommend using it in the Hubs (or Links). Loading the different sources in the same Hub table creates load dependencies that are against the Data Vault philosophy. We also found that the performance improvement is not significant enough to violate Data Vault architecture principles. On the other hand, it should be noted that STRUCT or JSON attributes may be a very interesting modeling approach for the Staging Layer e.g. for the staging of JSON data feeds.

For a few specific cases, deviation from Data Vault modeling using STRUCT data types may provide benefits. For example, Nested and Repeated fields may be leveraged for Tightly-coupled or Immutable relationships like Session w/ Events or Order w/ Line Items and Infrequently changing data like Country, Region, Date, etc.

Partitioning: With a Data Vault modeling approach there is always one access path that is used in most of the queries. An example is filtering on the current valid record of a Satellite. To optimize this, many approaches use an additional END_DT or a CURRENT_FLAG column in Satellites. This can lead to the use of END_DT to create a partition to optimize the

access performance as shown in the following example.

```
CREATE OR REPLACE TABLE
dwh-stor-vault.raw.sat_customer
(
  CUSTOMER_PK BYTES
  ,CUSTOMER_HASHDIFF BYTES
  ,CUSTOMER_NAME STRING
  ,CUSTOMER_ADDRESS STRING
  ,EFFECTIVE_FROM TIMESTAMP
  ,LOAD_DT TIMESTAMP
  ,END_DT TIMESTAMP
  ,SOURCE STRING
)
PARTITION BY TIMESTAMP_TRUNC(END_DT, DAY)
```

It was observed that in this case, the current partition is predominantly the largest partition resulting in overhead negating any significant performance gain. Furthermore, the values in the END_DT columns need maintenance. This means that, besides the above simple INSERT statement for loading data, an additional MERGE statement needs to be used to insert the new effective records and update the existing END_DT values. However, BigQuery provides an unparalleled scan performance and this can be leveraged to minimize such complexities. So, for Data Vault we recommend employing an alternative partitioning scheme using a LOAD_DT column as demonstrated in the following example.

```
CREATE OR REPLACE TABLE
dwh-stor-vault.raw.sat_customer
(
  CUSTOMER_PK BYTES
  ,CUSTOMER_HASHDIFF BYTES
  ,CUSTOMER_NAME STRING
  ,CUSTOMER_ADDRESS STRING
  ,EFFECTIVE_FROM TIMESTAMP
  ,LOAD_DT TIMESTAMP
  ,SOURCE STRING
)
PARTITION BY TIMESTAMP_TRUNC(LOAD_DT, DAY)
```

Given that BigQuery automatically halves the cost of the data that has not been updated for 90 days, this scheme not only reduces complexity but also the overall cost. This results in about a 50% reduction in storage cost. This is because every new delta load is inserted into a new partition and does not change any existing partition. However, it should be noted that this scenario does not support logical delete handling e.g. deleted records in the source. If this is a requirement, an additional column needs to be introduced to capture this as we do not have an END_DT materialized.

For very large Satellites storing fact or event data where long-term history is less frequently queried, it is possible to offload the partitions to [Google Cloud Storage](#) (GCS) for additional cost saving. This is because BigQuery can access data in GCS using external tables. For example, it is possible to create a UNION VIEW to access the Satellite where VIEW is based on the UNION of a TABLE in BigQuery containing recent partitions and an EXTERNAL TABLE containing offloaded partitions.

Clustering: Data access is further optimized by using clustering in BigQuery. For example, if the data access pattern heavily relies on filters on CUSTOMER_NAME then we recommend clustering on CUSTOMER_NAME. Further, BigQuery does automatic re-clustering behind the scenes without slowing down running queries. Re-clustering is not charged to the user, unlike with competing products. So, clustering can not only increase the performance but significantly lower costs.

3.4. Business Vault

3.4.1 How to approach computation

In the Business Vault, we deal with Soft Business Rules, Data Quality, and Master Data Management. There are many reasons to refactor it over time, therefore we default to agility and implement transformations as views. The views are optimized only if a need for optimization arises.

When optimizing:

1. Before materializing, consider how often the data is queried compared to how often it has to be updated. Materializing a view may have no benefit if it has to be recalculated each time it is queried.
2. Storage costs are usually not a concern when materializing on BigQuery.
3. When materializing, use [Materialized Views](#) whenever possible, as they simplify data freshness management.
4. When deciding about partitioning and clustering, consider the query patterns and not how they are stored. If multiple access paths are needed, consider Materialized Views to implement multiple [\(re\)clustering](#) options.
5. Use tools like [Google Dataform](#) that make it easy to switch the materialization [technique](#) from views to tables.
6. Consider the optimization patterns available in Data Vault and briefly covered below.

Often part of the Business Vault has to be recomputed because of changes in the Business Soft Rules. Potentially, this additional computation load can impact the SLAs of other DWH workloads. This is not a problem on BigQuery. It can scale up to absorb the temporary extra load and then scale down. For customers using BigQuery [flat-rate pricing](#), if we anticipate a negative impact on SLAs of other workloads, we can decide to add some additional [Flex Slots](#)⁶ just for the duration of the recomputation. Of course, this can be automated.

3.4.2 Clear contracts

The final goal of the Business Vault is to provide data to the Information Marts. The tables or views that are exposed become a contract between the Business Vault developers and the Information Marts developers. We suggest not to directly expose a Business Vault entity (table, view, etc...) to the Information Marts. Instead, we recommend creating a simple one-to-one view and exposing this. The view

⁶ They would be assigned to project dwh-comp-vault if we decided to follow the recommendation of chapter 3.1

then can have a version in the name. For example, *hub_customer_v1*. BigQuery can automatically elide these views during the query planning so there is no performance penalty.

There are times that the Business Vault can evolve in such a way that it would break the contract made by one of these views. In this case, the Business Vault developer creates a new view (for example *hub_customer_v2*) and informs the Information Marts developers. If possible, the old view is adapted so that it can continue to work without breaking its contract and is assigned an end of life date. Information Marts developers have to switch to the new version before the end of life.

3.4.3 Development environments

Data Vault focuses on bringing agility to Data Warehouse. In this context, it is useful to go beyond the typical sequence of environments like development, staging, production, and add personal development environments. Using tools like [Dataform](#) where artifacts are stored in Git, it is easy for a developer to create a temporary personal development environment, deploy only the portion of the Data Vault of interest, complete the development, and then destroy the environment. This technique, borrowed from the DevOps practices, speeds up development and increases quality. The serverless nature of BigQuery makes this possible. In particular, [Table Snapshots](#) permit to get data from the production environment into the personal development environment without incurring in extra storage costs or copies.

3.4.4 Point-In-Time Tables

For Hubs and Links with multiple Satellites, you need to build a consistent vision at a given instant (snapshot time). For a given snapshot time, for each PK, for each Satellite, the correct `EFFECTIVE_FROM` value is selected. Point-In-Time Table (PIT Tables) is an optional Data Vault pattern that pre-computes this.

We suggest using [Wildcard Tables](#) and storing each snapshot on its own table. This makes the write path efficient and leaves partitioning and clustering available to optimize the read access.

```
CREATE OR REPLACE TABLE
dwh-stor-vault.business.pit_customer_20160501
(
  CUSTOMER_PK BYTES
  , CUSTOMER_KEY STRING
  , SNAPSHOT_DATE TIMESTAMP
  , SAT1_PK BYTES
  , SAT1_EFFECTIVE_FROM TIMESTAMP
  ...
  , SATN_PK BYTES
  , SATN_EFFECTIVE_FROM TIMESTAMP
)
PARTITION BY ...
```

In the example above, we store a foreign key (`SAT*_PK`) for each Satellite of the Hub. This permits joining the PIT table with the Satellite tables. Normally, the `SAT*_PK` is equal to the Hub's PK (`CUSTOMER_PK` in our example) unless the Satellite is missing this information (e.g. late arrival). In that case, it references a Ghost record in the Satellite table. With Ghost Records in the Satellites, we avoid NULL values in the `SAT*_PK` columns. This permits the use of an `INNER JOIN` when joining the PIT Table and the Satellites. This is good also because Materialized Views support `INNER JOIN`. In case you don't care about `INNER JOIN`, you can use `OUTER JOIN` and avoid `SAT*_PK` columns and Ghost records.

3.4.5 Extended PIT Tables

We can go further with materialization by precomputing the PIT / Satellites joins in an Extended PIT Table. This results in a very wide table that is a problem for row-oriented databases but is completely fine for BigQuery. In the following example, we built an Extended PIT and put each Satellite in a dedicated `STRUCT`. The `STRUCT` is useful to avoid column names clashes and consequent name mangling.

```
INSERT INTO dwh-stor-vault.business.epit_customer
(
  SELECT
    pit.CUSTOMER_PK
  ,pit.CUSTOMER_KEY
  ,pit.SNAPSHOT_DATE
  ,(SELECT AS STRUCT sat1.* EXCEPT
    (SAT1_PK,LOAD_DT,SOURCE)) sat1
FROM dwh-stor-vault.business.pit_customer pit
INNER JOIN dwh-stor-vault.business.sat_1 sat1
ON (pit.PASSENGER_PK = sat1.SAT1_PK AND
    pit.SAT1_EFFECTIVE_FROM = sat1.EFFECTIVE_FROM)
...
)
```

3.4.6 Bridge Tables

A Bridge Table is another optional Data Vault pattern meant for performance optimization. A Bridge Table is built in snapshots like the PIT Table. Similarly, we suggest the same techniques discussed for the PIT Tables.

3.5. Information Marts

3.5.1 Access controls

When a user accesses data, BigQuery requires that the identity has the proper permissions to use the resource. You can grant permissions to a user, a group, or a service account on any level of the resource hierarchy.

We suggest using one GCP project per Information Mart by default, as described earlier. This allows us to manage permissions for our marts on the following levels:

Project. Manage access here for users that should have access to all database objects related to one Information Mart.

Dataset. A dataset is a collection of tables, views, ML models, and other database objects. You can have many datasets in a project. Use this to assign more granular permissions inside your mart.

Table/View. Give users access to individual tables and views if required. You may also use an [authorized view](#) to give more granular access to a table.

Column-level security. BigQuery provides fine-grained access to sensitive columns using policy tags, or type-based classification, of data. Use Data Catalog to [create and manage a taxonomy](#) and policy tags for your data. You can also use the GCP Data Loss Prevention API to automatically detect sensitive data, like PII, in your tables and restrict access to those columns.

Row-level security. BigQuery supports row-level security like traditional databases.

```
CREATE ROW ACCESS POLICY emea_filter
ON dwh-mart-1.sales_forecast
GRANT TO ("group:sales-emea@example.com")
FILTER USING (SALES_REGION = "EMEA")
```

[Row-Level Security](#) is a powerful feature. However, it may impact performance. For example, Row-level Security may cause a query to bypass the materialized view and BI Engine cache layers. This would force BigQuery to re-compute the results for every query rather than using the cache. Hence authorized views may be preferable in some situations where row-level filtering is required.

3.5.2 Billing & chargeback

Just like access controls, billing attribution for BigQuery (and Google Cloud Platform) usage can happen on different levels.

- BigQuery **storage cost** will always be billed to the project that contains the dataset. Using one project per mart allows you to break down storage costs by mart on the GCP invoice.
- **Query cost** will be billed to the Origin Project of a query, which may or may not be the same project that is used for storage. This allows different teams to query the same mart using different Origin Projects and makes sure each team can be billed internally for their usage. If you need more granular insights into usage, [labels](#) can be attached to datasets and queries.

3.5.3 Modeling the Information Mart

BigQuery does not prescribe a particular physical or logical model for Information Marts and supports all patterns typically used with Data Vault. Let's highlight some patterns that we have seen.

Logical structure: Some customers use heavily denormalized schemas, others prefer traditional modeling based on star schema, and many use both. For the following, we assume a standard star schema to illustrate the key points.

Physical structure: The Information Marts can always be regenerated from Raw and Business Vault layers. For example, switching from view to table (for performance reasons), or switching from a 3NF to a denormalized schema can be done at any time. This allows Data Vault users to flexibly respond to any emerging needs of your business users.

Virtualization using views. Data Vault implies a lot of joins between Hubs, Links, and Satellites. Depending on data volumes, virtualized Information Marts may provide good performance. This is especially true if wide PIT and Bridge Tables are used. As mentioned above, BigQuery's columnar architecture and cheap storage allow use of wide PIT/Bridge Tables, pre-joining Links, Hubs, and Satellites economically with good

performance. However, usually, you will want to materialize for best performance. Let's look at the two main options.

Standard tables. Google [Dataform](#) (or another ELT tool of your choice) is a convenient way to create Information Mart tables and keep them in sync with the lower vault layers. The dependency resolver in Dataform will make sure all derived tables are updated whenever a source table is updated. Additionally, Dataform makes it easy to switch between views and tables just by changing [a configuration option](#).

Materialized views. These are even more convenient because BigQuery manages the materialization and guarantees that queries will never see stale data. For example, the Raw Vault contains data at the original granularity and it is common to use different grain in the Business Vault or Information Marts. Materialized views are useful for [pre-aggregating data before](#) or [during](#) a demanding join to reduce the grain of tables⁷. Like indexes can be used to speed up lookups, materialized views can be used to boost analytic queries by [pre-filtering](#), [pre-computing](#), or [re-clustering](#) a table. Indeed, the BigQuery query planner transparently looks for materialized views that already contain the needed data in order to skip their computation.

External tables. An Information Mart may use tables that are not even stored in BigQuery native storage. Big Data [formats](#) like Parquet in Google Cloud Storage may be queried thanks to federated queries as outlined above. Another use case would be [storing reference data](#) in [Google Sheets](#): a spreadsheet allows business users to easily update the data any time, while BigQuery can join the always up-to-date sheet to other tables just like a regular table.

⁷ This highlights another advantage of using PIT tables in the Business Vault: as they allow us to use inner joins instead of outer joins for dimensions, they enable us to use materialized views in our Information Mart layer

4. Conclusions to set up a Data Vault model on BigQuery

This paper described the general principles of Data Vault and how they can be applied to an implementation of a DWH using BigQuery. We addressed the various layers within the Data Vault architecture and we found that, in general, there are few specific rules to consider using BigQuery versus other relational databases. Being an ANSI-compliant relational database, BigQuery users can easily start Data Vault projects in very much the same way as they used to. Only some specifics around running a cloud-native database, e.g. addressing security and folder structures, need to be addressed.

Often, distributed databases have problems handling complex data models with a lot of joins, as joins create an overhead for accessing the data. BigQuery addresses this by dynamically changing the execution plan during processing and not just relying on initial statistics. BigQuery redistributes data if data is found to be skewed whilst processing it. The combination of these features makes BigQuery an ideal foundation for a large-scale Data Vault implementation.

We recommend relying heavily on the cost-effective storage of BigQuery to improve access performance. BigQuery is based on a dedicated cloud storage system, making it very cost effective to store data. So for tuning a Data Vault implementation, we recommend the use of persisting support structures like PIT tables.

One of the key aspects of Data Vault projects is the focus on an agile and incremental approach to building the DWH. BigQuery further enriches this by providing a fully serverless architecture. Database administrator work is significantly reduced. There is no nodes management, indexes management, manual distribution, etc. This allows the user, together with new modern approaches for ELT automation like Dataform, to fully concentrate on the business logic instead of the technical constraints of the solution, further completing the vision of Data Vault.

Data Vault 2.0 on Google Cloud BigQuery

July 2022

Interested in getting started? [Contact us](#) to learn more.