Data Vault Partitioning Strategies
WHITE PAPER
CONTENTS
1 Introduction .......................................................................................................................... 3
2 Data Vault Modeling ............................................................................................................ 4
  2.1 What is Data Vault Modeling? ......................................................................................... 4
  2.2 Hubs, Links and Satellites ............................................................................................... 4
  2.3 Technical Columns .......................................................................................................... 5
  2.4 Primary Key, Unique and Foreign Key Constraints ......................................................... 5
  2.5 Example of Data Vault Model ......................................................................................... 6
3 Partitioning .......................................................................................................................... 7
  3.1 Partitioning Features ....................................................................................................... 7
  3.2 Partitioning Types ............................................................................................................ 8
4 Strategy 1: Partitioning by Load Date .................................................................................. 9
  4.1 Example .......................................................................................................................... 9
  4.2 Usage of Partitioning Features ....................................................................................... 10
  4.3 Pitfalls ............................................................................................................................ 11
5 Strategy 2: Partitioning by Load End Date ......................................................................... 12
  5.1 Example .......................................................................................................................... 12
  5.2 Usage of Partitioning Features ....................................................................................... 13
  5.3 Pitfalls ............................................................................................................................ 14
6 Strategy 3: Partitioning by Hub Key ................................................................................... 15
  6.1 Example .......................................................................................................................... 15
  6.2 Usage of Partitioning Features ....................................................................................... 17
  6.3 Pitfalls ............................................................................................................................ 17
7 Summary ............................................................................................................................. 18
1 INTRODUCTION

Partitioning is a database feature that splits large tables into multiple physical units, called partitions. This has several advantages for query performance, database administration and information lifecycle management. Partitioning is often used in data warehouses, for example for fact tables in a star schema.

But how can partitioned tables help to improve performance in a Data Vault environment? While loading data into Data Vault tables is usually easy and fast, it is a challenge in many projects to extract information from an extensive Data Vault schema. Is it possible to use partitioned tables to improve these extraction queries? What partitioning methods and partition keys are useful for this purpose? Are there other advantages when Data Vault tables are partitioned?

This white paper suggests three partitioning strategies that can be helpful for Data Vault schemas. Each of these strategies has some benefits, but may not be feasible in all situations. Also, there are some pitfalls and implementation details to know for each strategy.

The purpose of this document is to give an overview of different partitioning strategies for Data Vault tables and to explain their advantages and challenges. After reading this white paper you should be able to decide which strategy fits best for your specific project environment.

The examples in this white paper are explained with Oracle Database 12c, but they can also be implemented with other database systems that support partitioned tables. Eventually, some implementation details must be adapted for other systems.
2 DATA VAULT MODELING

The following sections give an overview of the base concepts of Data Vault Modeling. They are required to understand this white paper. If you are familiar with Data Vault Modeling, you can skip this chapter.

2.1 What is Data Vault Modeling?

Data Vault Modeling is a database modeling method, especially designed for data warehouses with a high number of structure changes. The basic concept of Data Vault is to split information in a way that allows easy integration and historization of the data. Additionally, the model can be enhanced without migration of existing tables. With three types of tables – Hubs, Links and Satellites – comprehensive and extensible data models can be built.

Data Vault Modeling is typically used for modeling the Core layer of a data warehouse or to build an Enterprise Data Warehouse (EDW) with many different source systems. BI users do not access the Data Vault tables directly, but run their queries and reports on dimensional data marts that are loaded from the Data Vault layer.

2.2 Hubs, Links and Satellites

A Data Vault model consists of various table types. Each table type has a specific purpose and contains different kind of attributes.

<table>
<thead>
<tr>
<th>Table Type</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Hub</strong></td>
<td>The <em>Hub</em> table contains the business key (one or more attributes) of a business concept and is used to identify individual business entities. Additional attributes (except technical columns) or relationships are not allowed in a Hub.</td>
</tr>
<tr>
<td><strong>Link</strong></td>
<td>All relationships between Hubs are stored in <em>Link</em> tables. A Link contains two or more foreign key columns to Hubs, but no additional attributes except technical columns.</td>
</tr>
<tr>
<td><strong>Satellite</strong></td>
<td>All context attributes, i.e. descriptive attributes or measures of a business entity, are stored in <em>Satellite</em> tables. Each Satellite is attached to exactly one Hub (or to a Link, if Link Satellites are used in the specific Data Vault implementation). A Hub (or Link) may have multiple Satellites. Satellites contain the complete history of the data. For each source record, a new version is written to the Satellite when at least one attribute has changed in the source system.</td>
</tr>
</tbody>
</table>

In some implementations of Data Vault, additional table types are used. In this white paper, we focus on the three main table types.
2.3 Technical Columns

For each table type in Data Vault, there are several technical columns required to complete the table design. They are described here because they are candidates for partition keys in the following chapters.

<table>
<thead>
<tr>
<th>Column</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Technical Key</strong></td>
<td>For Hubs and Links, a technical key is used as the primary key column of each row. This can be either a sequence number or a hash key built on the business key column(s). The technical keys are also used as foreign key columns on the referring Links and Satellites.</td>
</tr>
<tr>
<td><strong>Load Date</strong></td>
<td>Each row in Data Vault contains a load date (or load timestamp). It contains the point in time when the record was inserted in the Data Vault. Caution: This date is not related to any business validity date. In Satellites, the load date is part of the primary key.</td>
</tr>
<tr>
<td><strong>Load End Date</strong></td>
<td>In some implementations of Data Vault, a load end date is stored in the Satellites to improve performance of extraction queries. Whenever a new version of a record is loaded, the load end date of the previous version is updated to the load date of the next version.</td>
</tr>
<tr>
<td><strong>Record Source</strong></td>
<td>To track the origin of every record, each row in Data Vault contains a record source column. It contains the information from which source the record was loaded. This should be at least the source system, eventually the specific source table where data is loaded from.</td>
</tr>
</tbody>
</table>

Additional technical columns (e.g. extraction dates, ETL audit identifiers, hash difference attributes) can optionally be used, but are not relevant for this white paper.

2.4 Primary Key, Unique and Foreign Key Constraints

Because constraints are important for some of the partitioning strategies described in this white paper, here a short overview of the constraints defined for each table table.

<table>
<thead>
<tr>
<th>Table Type</th>
<th>Constraints</th>
</tr>
</thead>
</table>
| **Hub**    | Primary key on the technical key (called Hub Key in this document)  
Unique constraint on the business key column(s) |
| **Link**   | Primary key on the technical key  
Foreign keys to two or more Hubs, each containing the corresponding Hub Key |
| **Satellite** | Primary key on Hub Key and Load Date  
Foreign key to Hub, referring to corresponding Hub Key  
(for Link Satellites, the technical key of the Link is referred) |
2.5 Example of Data Vault Model

A Data Vault model usually contains a high number of tables. For this white paper, we use a very simple example with two Hubs, one Link and three Satellites.

![Data Vault Model Diagram]

The example is part of a Data Vault introduction video¹ and contains data for a craft beer brewery. Let’s assume this brewery is so big that it is worth to think about partitioning. To understand the partitioning strategies in this white paper, it is not required to understand how beer is brewed. The following explanations are just for information about the “business case”. The example in figure 1 contains two business concepts. For each of them, a Hub with one or two Satellites is created.

- Each row in the Hub H_BEER represents a specific kind of beer brewed by the microbrewery. Each beer has a unique name, which is used as the business key of the Hub. Two Satellites are attached to the Hub: S_BEER_DESCRIPTION² contains several attributes to describe the beer, e.g. beer style, alcohol by volume (ABV) or international bitterness units (IBU). The recipe to brew the beer is stored in a Satellite S_RECIPE. During the lifetime of a beer, its description as well as the recipe can be changed.

- In each brewing batch, called a brew, an amount of beer of a particular type is produced. Each brew has a unique brew number. This is the business key of the Hub H_BREW. For traceability and quality assurance, the detail information of each brew must be documented. This information is stored in the Satellite S_BREW_JOURNAL.

Although only one kind of beer is brewed in one brew, the relationship between a beer and a brew is implemented as a many-to-many relationship in Data Vault. The relationship is stored in the Link L_BEER_BREW.

---

¹ Introduction Video “How to create a Data Vault Model” [https://www.youtube.com/watch?v=Q1qj_LJeawc](https://www.youtube.com/watch?v=Q1qj_LJeawc)

² In some pictures of this document, this Satellite is named S_BEER, because the complete Satellite name is too long for the graphics
3 PARTITIONING

The following sections give an overview of partitioning. If you understand the reasons for partitioning in databases and know the various partition types, you can skip this chapter. The different partition strategies for Data Vault are described in the following chapters of this document.

3.1 Partitioning Features

Partitioning is the subdivision of a big table into multiple smaller parts, so called partitions. There are several reasons to create partitioned tables, mainly for performance improvements and information lifecycle management.

Figure 2: Benefits of a partitioned table

- **Partition Pruning:** When a query on a partitioned table contains a WHERE condition with a restriction on the partition key, only a subset of the partitions have to be read. This is often the case in data warehouses for tables with a huge amount of historical data. Most of the users are only interested in a subset of this information. For example, a table contains sales transactions of several years, stored in quarterly partitions. A user runs a report for a sales overview of the year 2013. To retrieve the result set, only the four quarterly partitions of this year must be read from the table, as shown in the example (1).

- **Partition-wise Join:** When two tables are partitioned on the same partition key (e.g. primary key on the master and foreign key on the detail table), a join of these tables can be performed for each pair of partitions separately. This is called full partition-wise join. If only one of the tables is partitioned, a partial partition-wise join can be used instead. The advantage of a partition-wise join is that only the rows of corresponding partitions are to be compared for the join, and that the individual join operations of all partitions can run in parallel.

- **Rolling History:** A data warehouse often contains historical data of several years. To avoid that the amount of data increases and the response times are getting slower, outdated data must be purged from time to time. When a table is partitioned on a date column, it is easy to drop old partitions and create new partitions. In the example (2), the oldest partitions Q1/2011 and Q2/2011 are dropped. The free disk space can be reused for the new partitions Q1/2015 and Q2/2015.

- **Physical Data Distribution:** Splitting a large table into several partitions also allows the controlled distribution of the data to several locations (disks, tablespaces or database files). This has advantages for I/O performance, but also allows to backup only the current partitions. In example (3), only the partition Q4/2014 must be saved. The historical data - if not changed - are already saved by previous backups. This method is suitable for large tables in which only new data sets are inserted.
Partition Exchange Loading: Instead of loading new data directly into a partitioned table, it is possible to load the data into an intermediate table with the same structure. This intermediate table is then exchanged with the corresponding partition of the target table. This method has advantages in load performance, repeatability of load jobs and availability of the data. In the example (4), the current partition Q4/2014 is loaded using partition exchange.

3.2 Partitioning Types

There are multiple methods available for partitioning a table. The following list gives an overview of the main partitioning types of the Oracle database. For other database systems, this may be different, but the base principles are similar.

<table>
<thead>
<tr>
<th>Partitioning Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>RANGE Partitioning</strong></td>
<td>Each partition contains a range of values of the partition key. This is typically used for DATE or TIMESTAMP columns. For example, it is possible to create monthly partitions with RANGE partitioning, based on a date key. An extension of this method is INTERVAL partitioning. It allows to create new partitions automatically as soon as the first key within a new range is inserted into the table. This reduces the overhead to maintain new partitions.</td>
</tr>
<tr>
<td><strong>LIST Partitioning</strong></td>
<td>Partitions are created for individual values or lists of values. It is also possible to create a default partition for all other values. Oracle 12c Release 2 introduces AUTOMATIC LIST partitioning. With this method, a new partition is created for each distinct value of the partition key.</td>
</tr>
<tr>
<td><strong>HASH Partitioning</strong></td>
<td>This method is used to distribute the data of a table to a number of partitions of (almost) the same size. In contrast to the other partitioning methods, no specific values or ranges are defined for the individual partitions. Only the number of partitions is specified. Oracle calculates a hash value for each partition key to derive in which partition it is stored. HASH partitioning is mainly provided for parallel queries or DML statements on the partitioned table and for partition-wise joins. The number of hash partitions should be a power of two, e.g. 2, 4, 8, 16, etc.</td>
</tr>
<tr>
<td><strong>Composite Partitioning</strong></td>
<td>Partitions can be subdivided into subpartitions. This is called “Composite Partitioning”. While only a few combinations were supported in older Oracle versions, all combinations of RANGE, LIST and HASH are now possible since Oracle 12c. For example, it is feasible to create a table with LIST partitions per country, each of them containing RANGE subpartitions per month.</td>
</tr>
</tbody>
</table>

Additional partitioning types, such as reference partitioning or virtual column-based partitioning are supported by Oracle, too. Because they are not relevant for this white paper, these special types are not described here.
4 STRATEGY 1: PARTITIONING BY LOAD DATE

Every table in Data Vault contains a “Load Date” column. Although it seems to be obvious to use this column as a partition key, this does not have the expected advantages in most situations. But for specific use cases, partition by load date may be a suitable strategy.

In Data Vault modeling, we do not differentiate between master data (“dimensions”) and transactional data (“facts”). For example, a “product” and a “sales transaction” are both business concepts with an own Hub and one or more Satellites. But for the physical implementation of the Data Vault tables, it is a useful information to know whether the data is frequently updated (typical for master data) or written once by an event (typical for transactional data).

For master data, every update creates a new version in a Satellite. For one business key in the Hub, there are typically multiple versions in each Satellite. Every version has a distinct load date.

For transactional data, only one row per business key is inserted in the Satellite. This is done at the same time when the Hub row is inserted. The same happens for the Links. When a sales transaction refers to a product, exactly one row per transaction is inserted in the Link between the Hub “Product” and the Hub “Sales transaction”. This can be helpful for partitioning.

4.1 Example

A brew in our craft beer brewery example is a kind of a transaction event. It happens on a specific date (in fact, brewing beer takes more than one day, but let’s make our example as simple as possible). When the brew is finished, the business key (a unique brew number), the brew journal and the reference to the beer are loaded into the Data Vault tables. This implies that for every brew, there is exactly one row in the Hub H_BREW, one row in the Satellite S_BREW_JOURNAL and one row in the Link L_BEER_BREW. All three rows have (almost) the same load date.

![Transaction Hubs, Links and Satellites are RANGE partitioned by load date](image)

Let’s assume our brewery is so big that it produces a very high number of brew batches. We decide to implement partitioning for the tables H_BREW, S_BREW_JOURNAL and L_BEER_BREW. All these tables are monthly partitioned by column LOAD_DATE.

This can be implemented with RANGE or INTERVAL partitioning, as shown in the following DDL statements:
CREATE TABLE H_BREW
( H_Brew_Key  RAW (16) NOT NULL,
  Brew_No     NUMBER( 4) NOT NULL,
  Load_Date   DATE NOT NULL,
  Record_Source VARCHAR2 (4 CHAR) NOT NULL
)
PARTITION BY RANGE (Load_Date) INTERVAL(numtoyminterval(1,'MONTH'))
(PARTITION p_old_data
  VALUES LESS THAN (TO_DATE('01-01-2015','dd-mm-yyyy')));

CREATE TABLE S_BREW_JOURNAL
( H_Brew_Key  RAW (16) NOT NULL,
  Load_Date   DATE NOT NULL,
  Brew_Date   DATE NOT NULL,
  Brewer      VARCHAR2 (40),
  ... 
  Record_Source VARCHAR2 (4 CHAR) NOT NULL
)
PARTITION BY RANGE (Load_Date) INTERVAL(numtoyminterval(1,'MONTH'))
(PARTITION p_old_data
  VALUES LESS THAN (TO_DATE('01-01-2015','dd-mm-yyyy')));

With INTERVAL partitioning, a new partition is created whenever a load date for a new month is inserted into the table. So, our tables H_BREW and S_BREW_JOURNAL will contain several monthly partitions.

For an incremental load of a Data Mart, we extract all brews of a particular load period (e.g. the last two months). For this kind of query, a filter on load date can be applied for Hub and Satellite table. This allows to read only the required partitions of both tables, i.e. partition pruning will be possible.

4.2 Usage of Partitioning Features

✓ Partition Pruning is the main purpose of this partitioning strategy. It works for queries with a filter on load date. This is useful for the incremental extraction of transactional data that is loaded once and never changed. For master data, this is typically not the case.

× Partition-wise Join is theoretically possible with this strategy, but not feasible. When a Hub and all its Satellites are loaded with the same load timestamp, a full partition-wise join between the Hub and each Satellite would be possible if the join condition is extended by the load date:

```
SELECT ...
FROM H_BREW h
JOIN S_BREW_JOURNAL s
  ON h.H_Brew_Key = s.H_Brew_Key
  AND h.Load_Date = s.Load_Date
```

To take advantage of this feature, it must be guaranteed that the load dates conform on both tables (i.e. data must be loaded in the same second), and there is never more than one version per Hub key available in the Satellite. Otherwise, not all versions would be selected.
White Paper Data Vault Partitioning Strategies

- **Rolling History** is another advantage of this partitioning strategy. Transactional data is eventually not needed anymore after several months or years. In this case, it is easy to drop old partitions with outdated information.

- **Physical Data Distribution** allows to move historical data to cheaper disk storage units and implement a backup strategy that saves only the newest partitions and omits partitions without any changes.

- **Partition Exchange Loading** is possible if the partitioning interval fits the load frequency. For example, with daily partitions and one load run per day, the data can be loaded into an intermediate table and then exchanged with the current partition of the target table. But there is no real benefit of this feature because the load jobs in Data Vault are fast and simple.

### 4.3 Pitfalls

In data warehouses, global indexes should be avoided. They complicate rolling history and cleanup tasks. When a partition is dropped, or exchanged, a global index is getting unusable and must be rebuilt. That's why we usually prefer local indexes on partitioned tables. But a local index cannot be created in all situations. A unique index can only be local when it contains the partition key. This is the case for the primary key index of a Satellite (column LOAD_DATE is part of the primary key). But the unique indexes that are created for the primary key and the unique constraint on a Hub will always be global indexes.

Fortunately, it is possible to avoid a complete index rebuild due to asynchronous global index maintenance – a new feature introduced with Oracle 12c. But for very large tables it would still be a better choice to create a local index - or no index at all.

One solution is to create only reliable constraints that are disabled:

```sql
ALTER TABLE H_BREW ADD CONSTRAINT H_BREW_PK
PRIMARY KEY (H_Brew_Key)
RELY DISABLE NOVALIDATE;

ALTER TABLE H_BREW ADD CONSTRAINT H_BREW_UN UNIQUE (Brew_No)
RELY DISABLE NOVALIDATE;

ALTER TABLE S_BREW_JOURNAL ADD CONSTRAINT S_BREW_JOURNAL_PK
PRIMARY KEY (H_Brew_Key, Load_Date)
RELY DISABLE NOVALIDATE;

ALTER TABLE S_BREW_JOURNAL ADD CONSTRAINT H_BREW_S_BREW_JOURNAL_FK
FOREIGN KEY (H_Brew_Key) REFERENCES H_BREW (H_Brew_Key)
RELY DISABLE NOVALIDATE;
```

In this case, the primary key and the unique constraint on the Hub are only declared, but not checked by the database. All corresponding constraints, including the foreign key constraints must be defined in the same way. This is a frequently used principle to define constraints in data warehouses. Because the uniqueness of the primary key and unique constraints is not checked anymore, no indexes will be created. So, there is no need to rebuild or maintain any global indexes. On the other hand, we must make sure with the load jobs that the uniqueness is always guaranteed.
5 STRATEGY 2: PARTITIONING BY LOAD END DATE

Most of the extraction queries in Data Vault retrieve only the current versions of the rows in each Satellite. The current versions are also important for load jobs into Satellites when a delta detection between new delivered data and already loaded data must be done. So, it is evident that identifying the current rows in a Satellite can be performed as fast as possible.

With the concept of partitioning, it would be very practical if all current rows of a Satellite are stored in a specific partition. Extraction jobs and delta detection have to read only this partition instead of the whole history stored in the Satellite.

The strategy described here can be useful for Satellites with many data changes. For example, if there are 20 different versions at an average stored per Hub key, only 5% of the rows in the Satellite are current versions. If these rows are stored in a “current partition”, while the rest of the table is stored in a “history partition”, it is very efficient to retrieve the current versions.

5.1 Example

Let's assume the beer recipes of our craft beer brewery are changed very often. So, we want to use the described partitioning strategy for the Satellite S_RECIPE in our example data model. To identify the current versions, a precondition is an additional column in the Satellite table, either a “load end date” or a “current flag”. For this example, we use the “load end date” – a solution that is used in many Data Vault projects.

The column LOAD_END_DATE of the last version must be updated when a new version for the same Hub key is inserted. A good practice is to set it to the same value as the LOAD_DATE of the new row (called close-open interval). The current row contains either NULL for the load end date, or a date in the future. In our example, we use the date 31-DEC-9999.

```sql
CREATE TABLE S_RECIPE
(
  H_Beer_Key     RAW (16)  NOT NULL,
  Load_Date      DATE NOT NULL,
  Load_End_Date  DATE DEFAULT ON NULL
                  TO_DATE('31-12-9999', 'dd-mm-yyyy'),
  Start_Temp     NUMBER (3),
  Mashing_Time_1 NUMBER (3),
  Mashing_Temp_1 NUMBER (3),
  ...
  Record_Source  VARCHAR2 (4 CHAR) NOT NULL
)
ENABLE ROW MOVEMENT
PARTITION BY LIST (Load_End_Date)
(PARTITION p_current VALUES (TO_DATE('31-12-9999', 'dd-mm-yyyy'))
 ,PARTITION p_history VALUES (DEFAULT));
```

The DDL statement creates a Satellite table with two LIST partitions: the partition P_CURRENT for all current rows, and a default partition P_HISTORY for all other rows. Two facts must be considered, as shown in the example. First, the table must be extended with a column LOAD_END_DATE. Here we use the DEFAULT ON NULL clause of Oracle 12c to set the end date of the current rows, but this could also be handled as part of the ETL job to load the Satellite.
Second, the table must be created with the option ENABLE ROW MOVEMENT. This is required because an update of the partition key from 31-DEC-9999 to any other date causes a row movement to the P_HISTORy partition.

This partitioning strategy can be used for all Satellites, but not for Hubs and Links.

**Figure 4:** All Satellites contain a current and a history partition

### 5.2 Usage of Partitioning Features

- **Partition Pruning** takes place when the current versions in a partitioned Satellite are selected. This is usually the case for extraction jobs as well as for delta detection when loading new rows into a Satellite. For Satellites with many data changes, this can highly improve load and extract performance.
- **Partition-wise Join and Rolling History** are not possible with this strategy.
- **Physical Data Distribution** allows to store the history partitions of the Satellites to cheaper disk storage units and implement a backup strategy that saves only the newest partitions and omits partitions without any changes.
- **Partition Exchange Loading** is theoretically possible to load the new versions into the current partition. A load table with all new versions and all unchanged current versions is created. In a second step, the old versions are moved from the current to the history partition with an additional INSERT statement. Finally, the current partition is exchanged with the intermediate table.

**Figure 5:** Special implementation case with partition exchange

This implementation does not need the ENABLE ROW MOVEMENT clause, but the complexity of the load jobs is much higher than with a normal MERGE statement. It may be useful for Satellites where most of the versions are replaced with every load. But in most situations, this approach has no benefit.
5.3 Pitfalls

To distinguish between current rows and history rows in a Satellite, an additional column “load end date” is required. This column is used as the partition key of the Satellite table. This is the only column of the table that is updated by the ETL jobs to load the Data Vault tables. One important advantage of Data Vault – the “insert only strategy” – is not possible anymore for Satellites with a load end date.

When a new version is inserted, the load end date of the previous version must be updated. In this case, the row must be moved from the partition P_CURRENT to the partition P_HISTORY. This is only allowed when row movement is enabled for the table. Row movement has an impact on indexes, too. When a row is moved from one partition to another, the corresponding ROWID in every index must be updated. For tables with many indexes, this can cause new performance issues. Because we create only very few indexes in Data Vault (for Satellites usually only the primary key index is required), this is not a real problem.

Finally, this strategy works only for Satellite tables. Hubs and Links are not partitioned and do not profit by this approach.
6 STRATEGY 3: PARTITIONING BY HUB KEY

Queries to extract data from a Data Vault model typically contain many joins. An approach to improve performance is to run the queries with full partition-wise joins. A full partition-wise join is possible if two tables are partitioned by their common key, i.e. the primary key column of one and the foreign key column of the other table. In Data Vault, full partition-wise join can be performed between Hubs and Satellites as well as between Hubs and Links. The appropriate partitioning method is HASH partitioning on the Hub keys.

With this strategy, the Hubs are HASH partitioned by their primary key column, the Hub key. The Satellites are HASH partitioned by the foreign key column, which is the same Hub key as for the corresponding Hub. The number of partitions must be the same for both table types. This allows full partition-wise joins between all Satellites and their associated Hubs.

Because Links contain references to multiple Hubs, we need a different partitioning method. Composite HASH-HASH partitioning can be used for Links with references to two Hubs. The partition key is the foreign key to one Hub, the subpartition key the foreign key to the other Hub. The same method can be used for “Same as Links”, i.e. Links with two references to the same Hub.

For Links with more than two references to Hubs, only two foreign key columns can be used as (sub)partition keys. See pitfalls in section 6.3 for more details.

6.1 Example

In our brewery example, the Hub and two Satellites on the left are partitioned by the Hub key of H_BEER. The Hub on the right and its corresponding Satellite are partitioned on the Hub key of H_BREW. For the Link table, Composite HASH-HASH partitioning on both Hub keys will be used.

![Diagram](image)

*Figure 6: HASH partitioning on the Hub keys allows full partition-wise joins between the tables*
White Paper Data Vault Partitioning Strategies

The Hubs are partitioned by their primary key column with eight HASH partitions:

```
CREATE TABLE H_BEER
(
    H_Beer_Key RAW (16) NOT NULL,
    Beer_Name VARCHAR2 (40) NOT NULL,
    Load_Date DATE NOT NULL,
    Record_Source VARCHAR2 (4 CHAR) NOT NULL
)
PARTITION BY HASH (H_Beer_Key) PARTITIONS 8;
```

For the Satellites, the foreign key column to the Hub is used as partition key. The number of HASH partitions is the same as for the Hub:

```
CREATE TABLE S_BEER_DESCRIPTION
(
    H_Beer_Key RAW (16) NOT NULL,
    Load_Date DATE NOT NULL,
    Style VARCHAR2 (40),
    ABV NUMBER (3,1),
    IBU NUMBER (3),
    Seasonal VARCHAR2 (10),
    Label_Color VARCHAR2 (10),
    Record_Source VARCHAR2 (4 CHAR) NOT NULL
)
PARTITION BY HASH (H_Beer_Key) PARTITIONS 8;
```

For Links, two foreign key columns to the Hubs are used for partitioning. In our example, the Hub key to H_BEER is used to create eight partitions. Each of them contains eight subpartitions, partitioned by the Hub key of H_BREW:

```
CREATE TABLE L_BEER_BREW
(
    L_Beer_Brew_Key RAW (16) NOT NULL,
    H_Beer_Key RAW (16) NOT NULL,
    H_Brew_Key RAW (16) NOT NULL,
    Load_Date DATE NOT NULL,
    Record_Source VARCHAR2 (4 CHAR) NOT NULL
)
PARTITION BY HASH (H_Beer_Key)
    SUBPARTITION BY HASH (H_Brew_Key)
    SUBPARTITIONS 8
PARTITIONS 8;
```

By the way: Although HASH partitioning is used for this partitioning strategy, this has nothing to do with the hash keys of Data Vault. In our example, the datatype of all the keys is RAW(16) because an MD5 hash is used. But the partitioning strategy works as well for numeric keys, generated by a sequence.
6.2 Usage of Partitioning Features

✓ **Partition-wise Join and Physical Data Distribution** are the main advantages of this partitioning strategy. The purpose of HASH partitioning is to distribute the data into a fix number of partitions, each of them having about the same number of rows. When joining a Hub with a Satellite or a Link with a Hub, each pair of partitions can be joined separately. The best benefit of this is when the SQL statements are executed in parallel mode.

The following listing shows the execution plan of an extraction query that joins the Link with the two Hubs and three Satellites of our brewery example model. The query is executed as parallel query:

<table>
<thead>
<tr>
<th>Id</th>
<th>Operation</th>
<th>S</th>
<th>PartX</th>
<th>PartY</th>
<th>TX</th>
<th>IN-OUT</th>
<th>PQ Distrib</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>SELECT STATEMENT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>PX COORDINATOR</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>PX SEND QC (RANDOM)</td>
<td>:TQ10003</td>
<td></td>
<td></td>
<td>0,03</td>
<td>P-&gt;S</td>
<td>QC (RAND)</td>
</tr>
<tr>
<td>3</td>
<td>HASH JOIN</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>PART JOIN FILTER CREATE</td>
<td>:BF0000</td>
<td></td>
<td></td>
<td>0,03</td>
<td>PCPN</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>PX RECEIVE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>PX SEND PARTITION (KEY)</td>
<td>:TQ10002</td>
<td></td>
<td></td>
<td>0,02</td>
<td>P-&gt;P</td>
<td>PART (KEY)</td>
</tr>
<tr>
<td>7</td>
<td>HASH JOIN</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>PART JOIN FILTER CREATE</td>
<td>:BF0001</td>
<td></td>
<td></td>
<td>0,02</td>
<td>PCPN</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>PX RECEIVE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>PX SEND PARTITION (KEY)</td>
<td>:TQ10001</td>
<td></td>
<td></td>
<td>0,01</td>
<td>P-&gt;P</td>
<td>PART (KEY)</td>
</tr>
<tr>
<td>11</td>
<td>HASH JOIN</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>PART JOIN FILTER CREATE</td>
<td>:BF0002</td>
<td></td>
<td></td>
<td>0,01</td>
<td>PCPN</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>PX RECEIVE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>PX SEND BROADCAST</td>
<td>:TQ10000</td>
<td></td>
<td></td>
<td>0,00</td>
<td>P-&gt;P</td>
<td>BROADCAST</td>
</tr>
<tr>
<td>15</td>
<td>PX PARTITION BASE ALL</td>
<td>1</td>
<td></td>
<td></td>
<td>0,00</td>
<td>PCPC</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>HASH JOIN</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>HASH JOIN</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>TABLE ACCESS FULL</td>
<td>S_BEAR</td>
<td>1</td>
<td></td>
<td>0,00</td>
<td>PCPN</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>TABLE ACCESS FULL</td>
<td>S_BEAR_DESCRIPTION</td>
<td>1</td>
<td></td>
<td>0,00</td>
<td>PCPN</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>TABLE ACCESS FULL</td>
<td>S_RECIPE</td>
<td>1</td>
<td></td>
<td>0,00</td>
<td>PCPN</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>PX PARTITION BASE ALL</td>
<td>1</td>
<td></td>
<td>0,01</td>
<td>PCPC</td>
<td></td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>TABLE ACCESS FULL</td>
<td>L_BEAR_BREW</td>
<td>1</td>
<td></td>
<td>0,01</td>
<td>PCPC</td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>PX PARTITION HASH JOIN-FILTER</td>
<td>:BF0001: :BF0001</td>
<td>0,02</td>
<td>PCPC</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>TABLE ACCESS FULL</td>
<td>S_BEAR</td>
<td>:BF0001: :BF0001</td>
<td>0,02</td>
<td>PCPN</td>
<td></td>
<td></td>
</tr>
<tr>
<td>25</td>
<td>PX PARTITION HASH JOIN-FILTER</td>
<td>:BF0000 :BF0000</td>
<td>0,03</td>
<td>PCPC</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>TABLE ACCESS FULL</td>
<td>S_BEAR_JOURNAL</td>
<td>:BF0000 :BF0000</td>
<td>0,03</td>
<td>PCPN</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

× **Partition Pruning, Rolling History and Partition Exchange Loading** are not possible with this strategy.

6.3 Pitfalls

For Links with more than two foreign keys to Hubs, only two of them can be used as a partition or subpartition key. The decision which two Hub keys should be used, it is recommended to compare the size (number of rows) of the Hubs. The Hub keys of the two largest Hubs referenced by the Link are then used as partition respective subpartition key for the Link table.

If Satellites are attached to a Link (e.g. to store the validities of the relationships), they will not have a Hub key. The foreign key of a Link Satellite is the Link key, a surrogate key that is defined on the Link table. In this case, it may be useful to use the Link key as partition key, and the Hub key of the largest referenced Hub as the subpartition key:

```
CREATE TABLE L_BEAR_BREW
  ( ... )
PARTITION BY HASH (L_BEAR_BREW Key)
  SUBPARTITION BY HASH (H_BEAR Key)
  SUBPARTITIONS 8
PARTITIONS 8;
```
7 SUMMARY

All three partitioning strategies in this white paper are useful in a Data Vault schema. But not all of them are suitable in all situations. While strategy 1 works only for transactional data, strategy 2 can only be implemented for Satellites. Strategy 3 is feasible for almost all Data Vault tables, with some restrictions on Links with more than two relationships.

Not all partitioning features are possible with each strategy. The following table gives an overview what feature is supported in which strategy. The details are described in the previous chapters.

<table>
<thead>
<tr>
<th>Partitioning Feature</th>
<th>Strategy 1(^3)</th>
<th>Strategy 2(^4)</th>
<th>Strategy 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Partition Pruning</td>
<td>✓</td>
<td>✓</td>
<td>×</td>
</tr>
<tr>
<td>Partition-wise Join</td>
<td>×</td>
<td>×</td>
<td>✓</td>
</tr>
<tr>
<td>Rolling History</td>
<td>✓</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>Physical Data Distribution</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Partition Exchange Loading</td>
<td>✓</td>
<td>✓</td>
<td>×</td>
</tr>
</tbody>
</table>

As you can see from this overview, there is no perfect strategy that fits all cases. Instead, each strategy has some benefits and some restrictions. The decision what partitioning strategy should be used in a specific Data Vault project depends on the amount and type of data, the frequency of changes and the complexity of the queries to extract information from the Data Vault schema. The explanations in this document hopefully help you to decide which strategy is the best for your particular needs.

\(^3\) only for transactional Hubs, Links and Satellites
\(^4\) only for Satellites