Institute for Parallel and Distributed Systems
Universität Stuttgart
Universitätsstraße 38
70569 Stuttgart
Germany

Master Thesis

Integration of Heterogeneous Data in the Data Vault Model

Geethanjali Nataraj

Study Program:	Computer Science
Examiner:	Dr. Holger Schwarz
Advisor:	Corinna Giebler, M.Sc. Informatik
Start Date:	17. July 2018
End Date:	17. January 2019

Abstract

Data increases tremendously with respect to volume, velocity, and variety. Nowadays, most of these data are unstructured like text documents, images, videos, Internet of Things (IoT) data, etc. Especially in enterprises, the analysis of semi-structured and unstructured data together with traditional structured data can add value. For example, semi-structured email data can be combined with structured customer data to keep a complete record of all customer information. Likewise, unstructured IoT data can be combined with structured machine data to enable predictive maintenance. Thereby, heterogeneous data need to be efficiently stored, integrated, and analyzed to derive useful business insights.

The traditional modeling techniques like Kimball's approach and Inmon's approach are primarily focused on modeling structured data. Due to vast amounts of data being collected and agile project execution, scalability and flexibility become more essential characteristics in data modeling. However, especially regarding flexibility, the traditional data modeling approaches used in data warehousing face some limitations. Therefore, Data Vault modeling was developed to overcome these limitations. However, the Data Vault model was designed for structured data. To combine these structured data with semi-structured and unstructured data, the Data Vault model therefore needs to be adapted. However, there exists no comprehensive approach to do so for both semi-structured and unstructured data.

This thesis, therefore, focuses on developing various modeling approaches to integrate semi-structured and unstructured data along with structured data into the Data Vault model. To this end, multiple use cases from different areas like Customer Relationship Management (CRM), Manufacturing, and Autonomous Car Testing that produce and use heterogeneous data are taken into consideration. Using examples from these areas, the different approaches are implemented and their advantages and disadvantages are discussed. In addition, the developed concepts are evaluated to check whether they fulfill the Data Vault characteristics.

Content

List of Figures

List of Tables

List of Acronyms

ERP	Enterprise Resource Planning
CRM	Customer Relationship Management
MES	Manufacturing Execution Systems
IoT	Internet of Things
3NF	Third Normal Form
ER	Entity Relationship
S3	Semi-Structured Schema
CM	Conceptual-Model
EER	Extended Entity Relationship
ORA-SS	Object-Relationship-Attribute model for Semi-Structured data
NLP	Natural Language Processing
RDF	Resource Description Framework
OWL	Web Ontology Language
DeepDWH	Deep Data Warehouse
CMS	Content Management System
CAD	Computer Aided Design
IMU	Inertial Measurement Units
LIDAR	Light Detection and Ranging
RADAR	Radio Detection and Ranging
GPS	Global Position System
EDW	Enterprise Data Warehouse
DV	Data Vault
BK	Business Key
VIN	Vehicle Identification Number
WMI	World Manufacturer Identifier
VDS	Vehicle Descriptor Section
VIS	Vehicle Identifier Section
RDBMS	Relational Database Management System

ESB	Enterprise Service Bus
PIT	Point in Time
OLAP	Online Analytical Processing
CMMI	Capability Maturity Model Integration
SHA-1	Secure Hash Algorithm-1
MD5	Message-Digest Algorithm
MPP	Massively Parallel Processing
JSON	JavaScript Object Notation
BSON	Binary JSON
XML	Extensible Markup Language
XSD	XML Schema Definition
DTD	Document Type Definition
URI	Uniform Resource Identifier
ID REF	ID Reference
CT	Complex Type
A	Attribute
TR	Translation Rule
AP	Approach
AL	Alternative
HDFS	Hadoop Distributed File System
CSV	Comma-Separated Values
HiveQL	Hive Query language
SQL	Structured Query Language
I/O	Input/Output

1 Introduction

Data are considered as a collection of information, which can be in the form of personal data, transactional data, web data, sensor data, etc. In recent days, the technology has increased tremendously and as a result, the data generated from various sources as well increases. Especially enterprises face the problem of heterogeneous sources, as data are produced by Enterprise Resource Planning (ERP) systems, CRM, Manufacturing Execution Systems (MES) and IoT. However, these data can be processed to identify useful business insights, such as customer preferences or behavior [MT16]. Thus, analyzing these data have more benefits.

Traditionally, the data were collected and organized in the data warehouse. Data Warehouses are considered to be a repository of historical data and are mainly for structured data. According to Bill Inmon, the data warehouse is defined as "a subject-oriented, integrated, time-variant and non-volatile collection of data to enable the decision-making process" [H02]. Data Warehouses can be modeled using various modeling techniques but Kimball's approach and Inmon's approach are the two widely used modeling techniques for data warehouses [B04] [PP12]. Kimball's approach is also known as dimensional modeling whereas Inmon's approach is also known as Third Normal Form (3NF). Fact table and dimensional table are the two key components of dimensional modeling like star schema. In 3NF model, the data are split into different entities as a relational table. Even though these data warehouse modeling techniques have been successful, they have limitations like reengineering, flexibility, and scalability. To overcome these limitations, Data Vault modeling was developed by Linstedt et al. [L016].

Data Vault model is a combination of star schema and 3NF. The main advantages of the Data Vault model are flexibility and scalability, because of which they are highly liked to be used in industry practices. However, through novel techniques, not only the amounts of data increases but also their complexity. The data coming from various sources are heterogeneous like structured, unstructured and semi-structured data. According to Gartner Group statistics, 80 percent of today's data are unstructured [LLY+11]. Therefore, in addition to structured data, the semi-structured and unstructured data also need to be stored and organized.

For a complete analysis of all enterprise data, the semi-structured and unstructured data need to be integrated along with the structured data. For example, data lakes can be used to store all these data together in one place. In order to understand these data, the data lake needs to be modeled [S14]. However, the Data Vault does not support the integration of semi-structured and unstructured data with structured data i.e., semi-structured and unstructured data that have to be integrated along with structured data in the Data Vault model, which is yet to be modeled.

1.1 Task

The Data Vault model is well suited for modeling structured data as there are some well-defined approaches to integrate structured data in the Data Vault. However, the question on how to integrate semi-structured and unstructured data into the Data Vault model without compromising its advantages is neither completely nor partially answered. To integrate semi-structured data into the Data Vault, there exist some approaches, but there are no approaches for unstructured data.

The aim of the work is to define a modeling concept to integrate semi-structured and unstructured data into the Data Vault model. To this end, suitable modeling techniques for semi-structured data will be discussed, and a new concept will be developed for both semi-structured and unstructured data. This concept is applied to multiple real-world use cases, which uses heterogeneous data. A prototype is implemented to prove the functionality of the concept, and later the concept is evaluated to check whether it still fulfills the Data Vault characteristics.

1.2 Structure

This thesis is segmented into the following chapters:

Section 2 - Related Work: This chapter discusses the various modeling techniques for structured, semi-structured and unstructured data. Additionally, some details about how to integrate unstructured data in the data warehouse are given.

Section 3 - Use Cases: This chapter discusses use case scenarios from the areas of CRM, Manufacturing and Autonomous Car Testing to show the need of integrating semi-structured and unstructured data into the Data Vault model.

Section 4 - Data Vault Model: This chapter discusses about the Data Vault with basic entities and the rules involved in the Data Vault modeling. Later describes the components of the Data Vault architecture and the comparison between the two existing versions of the Data Vault.

Section 5 - Existing Approaches to Integrate Semi-Structured Data in the Data Vault: This chapter discusses about the existing approaches to integrate JavaScript Object Notation (JSON) and Extensible Markup Language (XML) schema in the Data Vault.

Section 6 - Proposed Ideas to Integrate Semi-Structured and Unstructured Data in the Data Vault: This chapter presents a new approach on how to integrate semi-structured and unstructured data into the Data Vault model with different use cases like CRM, Manufacturing and Autonomous drive testing.

Section 7 - Implementation: This chapter shows the prototypical implementation of different approaches using various use cases.

Section 8 - Evaluation: This chapter compares the different approaches with their alternatives along with their advantages and disadvantages.

Section 9 - Summary and Future Work: This chapter discusses the outcome of this thesis and the possible future work.

2 Related Work

Data modeling is a communication tool for the business end users to present the information, which is captured. The goal of the data model is to fully and accurately represent all the data objects that are essential for the business. For instance, with the help of the data model, we can easily describe the key elements like tables, keys (primary and foreign keys), which are necessary for the design of the data structure for the structured data [BFG+06].

In this chapter, we will discuss about the various modeling techniques for structured data (in Subsection 2.1), semi-structured data (in Subsection 2.2), and unstructured data (in Subsection 2.3). Later in Subsection 2.4, we will discuss the integration of the unstructured data in data warehouses.

2.1 Modeling Techniques for Structured Data

Structured data has a well-defined format and is therefore typically organized in tables with a fixed set of columns. 3NF model, Dimensional modeling, and Data Vault model are some of the data modeling techniques for structured data based on the following references [PP12] [YL16] [KR13].

a) **3NF model** - Inmon pioneered the Corporate Information Factory (CIF) as integrated data warehouse architecture to represent the data using the 3NF modeling technique [KR13]. ER model is a high-level conceptual data model to simplify database design by capturing the relationship that exists between different entities. ER model was introduced by Chen in 1976 to model the data warehouse [C76].

   ER model is defined with the help of entities, relationships, and attributes as shown in Figure 2.1. The entity represents the real-world objects like *Product* or *Customer* and is denoted in a rectangle symbol. The relationship describes the connection that exists between two or more instances of one or more entities. For example, *Order* defines the relationship between the *Customer* entity and *Product* entity. The relationship is denoted by the diamond symbol. The attribute represents the properties that define the entities, and it is denoted by the oval symbol. For example, *Product Name* and *Product Price* are some of the attributes of the entity *Product* [KR13].

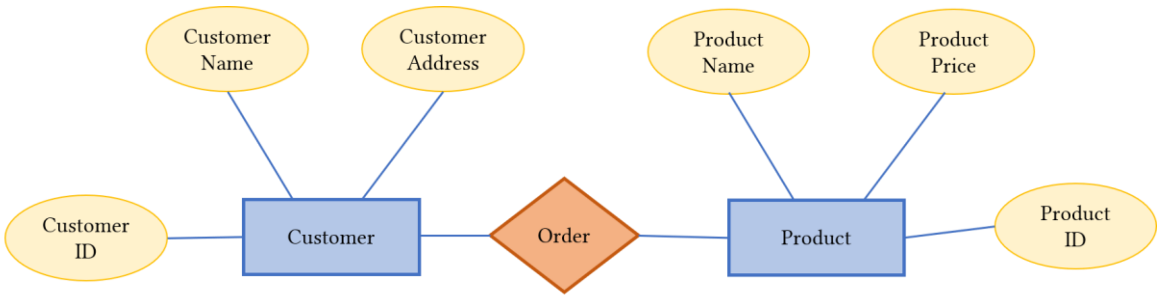


Figure 2.1: Example for ER Model

b) **Dimensional modeling** technique was introduced by Kimball to design the data warehouse. The degree of normalization is the major difference between 3NF model and dimensional model [KR13]. This modeling technique is proposed to improve the query performance when dealing with large queries, which is a major issue in 3NF [BFG+06].

Dimensional modeling consists of fact table and dimension table. Fact table stores the numerical measures about the particular business process like *Sales revenue* and the dimension table represents the facts that can be aggregated along the dimensions. For example, as shown in Figure 2.2, *Fact_Sales* represent the fact table with measures like *Sales*, *Product* and it stores the foreign keys (FK) like *Customer ID* and *Product ID*, which reference the primary keys of the dimensional tables. *Dim_Customer*, *Dim_Product*, etc. are the dimensional tables, which are connected to the fact table *Fact_Sales*. Each dimensional table like *Dim_Customer* has a primary key *Customer ID* and the descriptive information like *Customer Name*, *Customer Phone*. Dimensional model contains a large amount of data in the single fact table, and unlike the ER model, it contains more redundant data [BFG+06].

Star model, *snowflake model*, and *multi-star model* are the three types of dimensional models. Star schema comprises of one fact table and numerous dimensional tables, but the dimensional tables are not further divided (denormalized). If we further split the dimensional table of the star schema, it is known as snowflake schema. If multiple fact tables are combined with the help of dimension tables, then it is known as multi-star model [BFG+06].

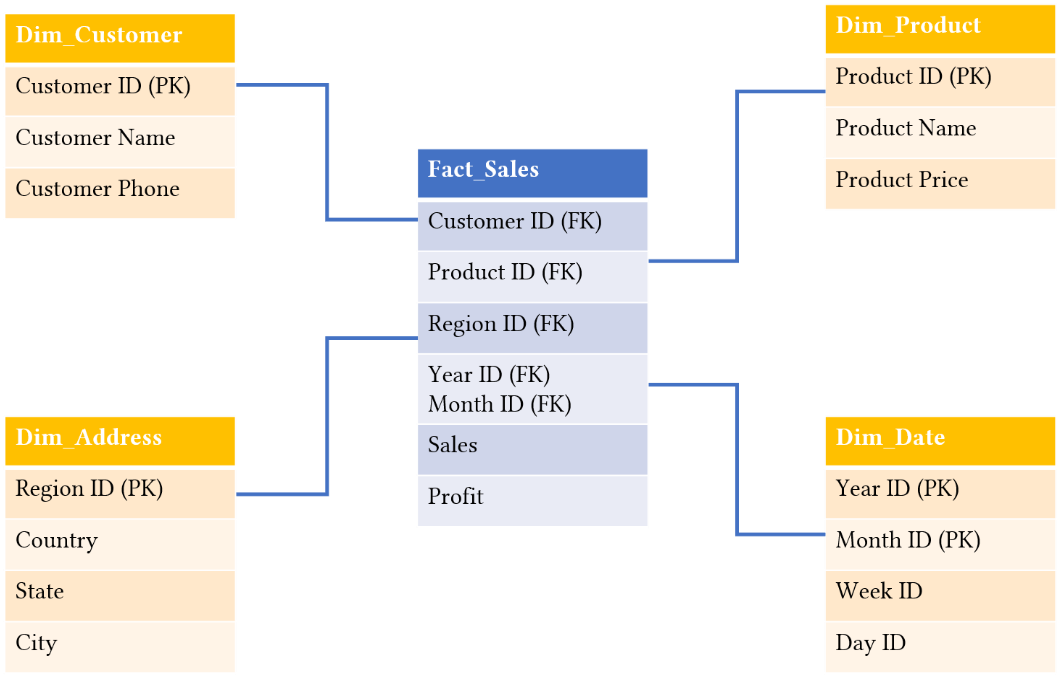


Figure 2.2: Example for Dimensional Model

c) **Data Vault model** - When a business requirement changes, both of these traditional data modeling techniques face limitations like traceability, scalability, and flexibility. In spite of the many solutions proposed to overcome these limitations especially for dimensional modeling, these solutions tend to decrease the data warehouse performance [NJ16]. To overcome all these challenges faced by traditional modeling techniques to implement the data warehouse in an agile way, the Data Vault modeling technique that accepts changes can be an alternative solution [NJ16] [BB13]. We will discuss in detail about the Data Vault modeling technique in chapter 4.

2.2 Modeling Techniques for Semi-Structured Data

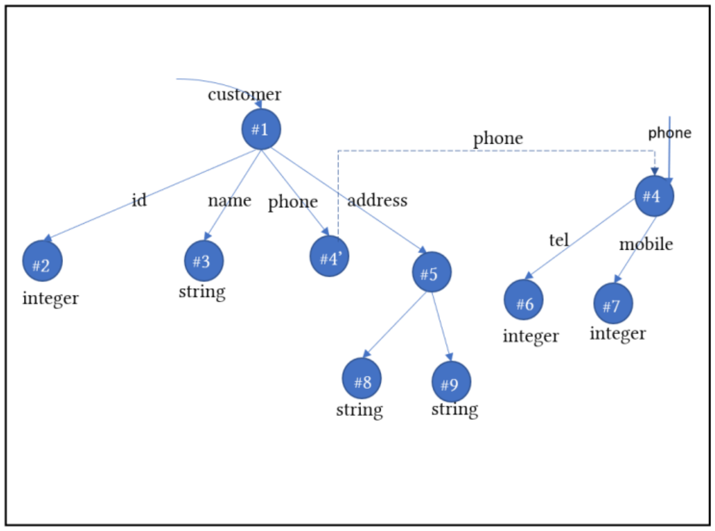
With the evolution of the Internet, semi-structured data has become popular. Semi-structured data has some structure, but they do not have a well-defined format to represent the data i.e., a fixed schema is not known in advance. Examples for semi-structured data are Extensible Markup Language (XML), JavaScript Object Notation (JSON), etc. There are various modeling techniques for semi-structured data like *Semi-Structured Schema graph (S3-Graph)*, *Conceptual-Model (CM) Hypergraph & Schema Tree*, *Extended Entity Relationship (EER) model & XGrammar* and *Object-Relationship-Attribute model* for Semi-Structured data (ORA-SS). We will now discuss S3-Graph, CM Hypergraph & Schema Tree, ORA-SS model in this Subsection based on [LWD+05].

a) According to [LLL+99], normal form for the **S3-Graph** was proposed and the data redundancy problem in the semi-structured database can be solved with the help of S3-Graph. It is defined as a directed graph with nodes and edges. Here, each node can be an entity node or a reference node. An entity node also known as leaf node represents an entity of atomic or complex datatype, whereas a reference node represents a node that refers to another entity node. Each edge has a tag to represent the relationship between the nodes and sometimes tag has a suffix '*' to signify that the element can have many child elements relationship. There are three types of edges like component edge, referencing edge and root edge. The component edge is represented by a solid arrow line whereas referencing edge is represented by a dashed arrow line. The root edge has no source node, and it is represented by a solid arrow line.

   S3-Graph illustrates the hierarchical structure of the elements, but it is difficult to differentiate the attributes of relationship sets and entity sets. With this graph, it is possible to represent binary relationship sets like 1:1 and 1: N relationship but it is not possible to express the ternary relationship sets (association among three different entities) [LWD+05]. As shown in Figure 2.3, *node #1* represents an entity node, which represents the entity *customer*; *node #3* represents an entity node, which holds a *string* representing the *name* of the customer. It means that any *name* entity will have the data of string type. The directed edge between *node #1* and *node #3* denotes that "each customer has at most one *name*". *Node #4'* represents a reference node, and it references the phone entity node. In this case, *node #4'* signifies the same entity *node #4*. The edge that connects *node #1* and *node #4'* signifies that "a *customer* has many *phone numbers*".

```
<customer details>
    <customer id="101">
        <name> Jim </customer name>
        <phone>
            <tel>12345 </tel>
            <mobile> 234515662 </mobile>
        </phone>
        <phone>
            <tel>82345 </ tel>
            <mobile> 674515563 </mobile>
        </phone>
        <address>
            <street> 12 abc street </street>
            <city> npmar </city>
        </address>
    </customer id>
<customer details>
```

a) XML



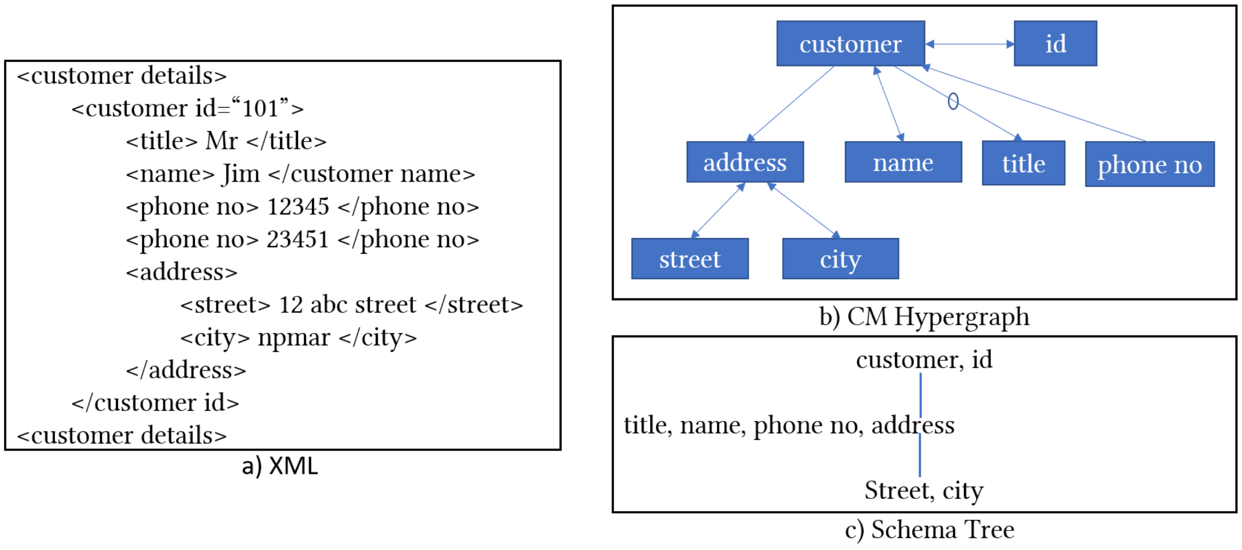
b) Schema for Semi-Structured Database

Figure 2.3: S3-Graph for an XML Document

b) Embley and Mok defined the **CM Hypergraph** and **Schema Tree** data model separately for designing a schema for semi-structured data in 2001 [LWD+05]. The CM Hypergraph consists of object sets/element sets denoted in a rectangle with a label. The participation constraints (N:N, N:1, 1:1, optional) of relationship sets are represented by the edges as follows:
   - N:N relationship – denoted by an edge without arrowheads
   - N:1 relationship –indicated by one arrowhead edge
   - 1:1 relationship – denoted by two arrowheads at both ends
   - Optional – denoted with symbol 'o' on the edge

In Figure 2.4, *customer*, *id*, *name*, etc. are object sets. The *customer* has a unique *id*, which is represented as 1:1 relationship. Likewise, the *customer* has one or more *phone numbers* represented as 1:N relationship. The edge between the *customer* and the *title* represents the relationship is optional. With the help of CM hypergraph, it is possible to express the binary and higher level (ternary, n-ary) relationships. In CM hypergraph the attributes and objects are not differentiated, the graph becomes complex.

In CM Hypergraph it is challenging to represent the hierarchical relationship, it is expressed with the help of the schema tree [LWD+05]. Schema tree is used to describe the hierarchical structure of the element sets, here the edges denote the relationship between element and sub-element. In Figure 2.4, *customer* and *id* are at the root; *name*, *phone no*, and *address* are nested within *customer*; *street* and *city* are nested within *address* forms the hierarchical structure.



```
<customer details>
        <customer id="101">
                <title> Mr </title>
                <name> Jim </customer name>
                <phone no> 12345 </phone no>
                <phone no> 23451 </phone no>
                <address>
                        <street> 12 abc street </street>
                        <city> npmar </city>
                </address>
        </customer id>
<customer details>
```

Figure 2.4: CM Hypergraph and Schema Tree for an XML Document

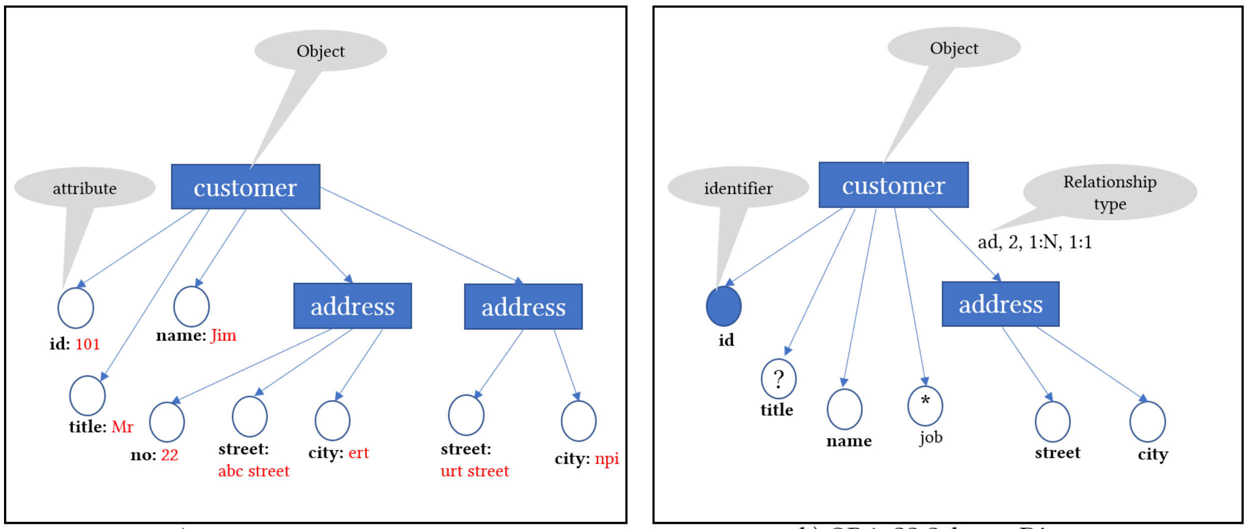
c) According to [WLL+02], it is difficult to represent the semantics for the current semi-structured databases. In order to represent the semantically rich data model mainly for semi-structured data, **ORA-SS** data model was introduced. The objects, attributes, and relationships are the three basic concepts of ORA-SS data model [DWL+00]. With the help of these concepts, four ORA-SS diagrams like instance diagram (similar to Document Object Model tree), schema diagram (similar to CM hypergraph), functional dependency diagram and inheritance diagram can be designed [DWL+00]. In ORA-SS data model, labeled rectangle denotes objects, and the labeled circle indicates attributes with its value.

ORA-SS instance diagram [LWD+05] was designed to show the instance and the difference among objects and attributes. It consists of internal and leaf nodes. Internal nodes represent the objects like *address* is signified with the labeled rectangle, whereas leaf nodes represent the attributes like *id*, *name*, *street* are denoted as labeled circles with a value *101*, *Jim*, *abc street* as shown in Figure 2.5 (a).

In ORA-SS schema diagram as shown in Figure 2.5 (b) was designed to differentiate among objects, attributes, and relationships. In addition to that, it shows the degree of the relationship, whether an attribute belongs to the relationship or an object. Attributes can be single-valued or multivalued, and it can be made mandatory or optional for the object class or relationship type as shown below [LWD+05]:

- The filled circle denotes that the attribute is a unique identifier (Example: *id*).
- If the circle contains '?', it means that attribute is optional and single-valued (Example: *title*).
- If the circle does not contain anything, then it means that attribute is mandatory and single-valued (Example: *name*, *street*).
- If the circle contains '*', then it means that attribute is optional and multi-valued. (Example: *Job*).

- If the circle contains '+', then it means that attribute is mandatory and multi-valued.
- The labeled rectangle represents the object class like the *customer*, and the relationship type between two object classes are defined by 'relationship name:degree of relationship:parent object class participant constraint:child object class participation constraint' (Example: *ad, 2, 1:N, 1:1*).
- Relationship name is *ad*, and the degree of relationship represents a binary relationship (*2*) between the two object classes *customer* and *address*.
- *1:N* is a parent object class participant, which defines that the parent object class *customer* can have one or more *addresses*
- *1:1* is a child object class participant, which specifies that the child object class *address* belongs to one and only *customer*.



a) ORA-SS Instance Diagram b) ORA-SS Schema Diagram

Figure 2.5: ORA-SS Instance and Schema Diagram

With these modeling techniques, we better understand the schema for semi-structured data like XML and JSON. However, they do not allow to integrate unstructured data and do not fit a data warehouse. Therefore, we need to develop new approaches to integrate the semi-structured and unstructured data into the Data Vault model, which will be discussed in chapter 6.

2.3 Modeling Techniques for Unstructured Data

Unstructured data like text in emails, text files, audio, video, images, IoT data, social media feeds do not have any structure i.e., it does not have a pre-defined data model. Hence, it is difficult to use it directly without knowing the schema. Unstructured data are classified as non-textual unstructured data and textual unstructured data [VNR14]. The unstructured textual data can be understood with the help of data modeling techniques like *Semantic Web* and *Ontology*, *Text Mining*, *Natural Language Processing (NLP)*, *Information Extraction models*, which we will discuss below in this Subsection.

a) **Text Mining** is used as a modeling technique in the paper [LKL+15] to develop the methodology to handle the unstructured data efficiently. It is used to extract the useful features from the text and to make the computers understand the meaning. This modeling is well suitable for textual unstructured data like a text document, pdf, etc. Data preparation, text processing, are some steps to perform text mining. Text processing involves sentence segmentation, tokenization, stop word removal and further processing like classification, association, etc.

b) **NLP** is used as a modeling technique in the paper [BH15] to automatically extract the information from unstructured data, which is stored in the natural language text. The main idea is to automatically generate the ER elements like entities, attributes and relationships from natural language specifications with the help of a Heuristics-based approach. According to this approach, entities are the nouns specified in the system requirements, and verbs are the relationships. Sentence segmentation, Tokenization, Tagged Parts of Speech, Chunking and Parsing are some of the processes involved in generating ER from NLP.

c) **Information Extraction** [GM16] is to automatically extract the structured information from the unstructured or semi-structured data sources. It is a challenging task to retrieve the information from textual unstructured data without changing its meaning. There are various techniques like Semantic Web to provide a solution to perform this challenging task [GM16]. Semantic web technology is an extension of World Wide Web (WWW), which is used to provide well-defined meaning for the data from various web sources, to make the machines to understand these data to deliver the exact answer for the web users request.

   Semantic Web uses the graph-based data model to store the data. Even though there are many techniques, Resource Description Framework (RDF) and Web Ontology Language (OWL) are the standard formats to represent the semantic data [QS13]. RDF is used to describe the different type of data that is available on the web with its own vocabularies. The basic building blocks of RDF are subject, predicate, and object, which are collectively known as RDF triples. RDF triples are denoted in the form of a graph where nodes represent the subject and object, whereas the directed edges represent the predicates (relationship). OWL is used to identify the nature of the resources and their relationships. OWL uses an ontology to express the semantic web information.

In the next Subsection, we will discuss the integration of unstructured data in the data warehouse.

2.4 Integration of Unstructured Data in the Data Warehouse

For complete analytics, the structured warehouse data need to be integrated with unstructured content to derive new insights. Integrating the semi-structured and unstructured data in a distinctive data warehouse is difficult because the traditional DWH is focused on structured data. A new form of data warehouse known as *Deep Data Warehouse (DeepDWH)* was introduced in

[GSM14] to provide flexible integration and enhancement of structured warehouse data and unstructured content.

Between unstructured content items and warehouse structured elements, the instance-level links are created. DeepDWH depends on these instance-level links, which are illustrated in a graph-based structure. From the simple graph, RDF graph and property graph, the property graph is used to illustrate the storage of non-DWH links. The reason is that the property graph model allows mapping a link information directly with the help of properties on edges [GSM14].

The unstructured data items stored in a Content Management System (CMS) and structured warehouse data stored in a Data Warehouse (DWH) are uniformly accessed in DeepDWH architecture.

Link-oriented concepts

Links can be created manually or generated automatically. However, generation of links automatically is an issue because there is no generic mechanism defined for automatic link extraction since the techniques are more dependent on the data source formats like images, text files, etc. This paper [GSM14] explains how to model and store the links, and also to define a unified link view to allow querying of data in the DeepDWH. The generic linking model represents the link approach, which is described at the meta-model level as shown in Figure 2.6.

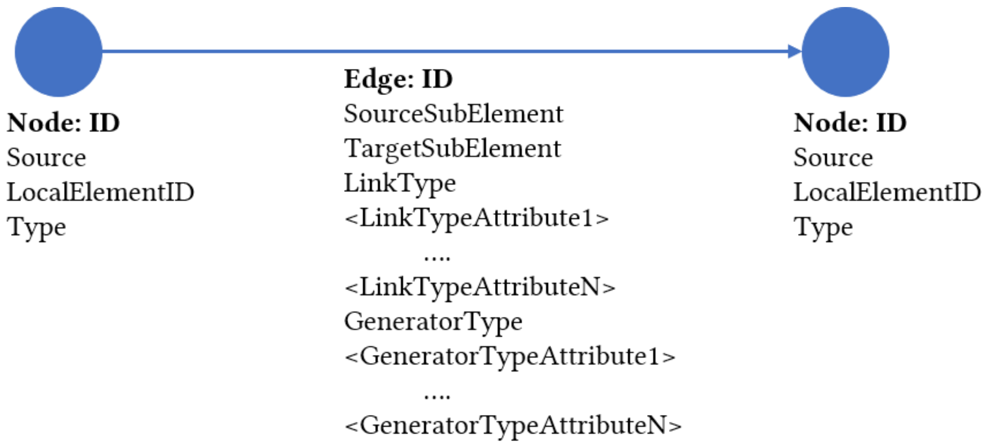


Figure 2.6: Generic Logical Schema for Links [GSM14]

A “*Node*” in the graph schema represents the element, which can be *structured DWH elements* or *unstructured content items*. The mandatory node properties shown in Figure 2.6 are as follows:

- The “*Source*” property is to identify the type of source system in which the respective element is available. The source can be either CMS for the content item or DWH for DWH element respectively.
- The property “*LocalElementID*” refers to the element identifier in the respective source system. LocalElementID for DWH element and content item are primary key attributes with its values and file object identifier respectively.
- The “*Type*” property refers to a special type of element. Type for DWH element and content item can be table name and text document respectively.

An "*Edge*" in the graph schema represents a link. Links are created by a "generator", and they are defined to be as directed, binary, typed and attributed relationship between elements [GSM14]. A link relates exactly two elements, where one element acts as a source, and other element acts as a target. The various link properties shown in Figure 2.6 are as follows:

- The properties like "*sourceSubElements*" and "*targetSubElements*" are optional and they are used to refine the relationship that exists between elements. In a DWH element, they denote the table attribute name whereas, in the content item, it refers to a specific file type like XQuery expression for an XML.
- A "*LinkType*" property represents the semantics of a link. For example, link types can be "Explains, IsExpertOn, refersTo, ExpressSentiment" [GSM14].
- A LinkType has various "*LinkType attributes*" to give additional information about a link. These link type and link type attributes are used to provide flexible relationship information.
- "*GeneratorType*" property specifies the link origin, i.e. whether a link is generated manually or automatically. Like link type attributes, the generator type also has generator type attributes.

If we model our DWH in the Data Vault, we want to keep the characteristics of the Data Vault. Therefore, we use the DeepDWH and the links as a base to investigate further and create modeling techniques in the Data Vault.

3 Use Cases

In this chapter, we will investigate some use cases that necessitate the integration of heterogeneous data. These use cases originate from the areas like CRM (Subchapter 3.1), Manufacturing (Subchapter 3.2) and Autonomous Car Testing (Subchapter 3.3). Problems related to the integration of heterogeneous data are discussed with the help of these use cases.

3.1 Customer Relationship Management

CRM is an enterprise approach that allows a business to improve the relationship with the customer, partners, and suppliers. The aim is to improve the business relationship with existing or potential customers. CRM collects data from various communication channels ranging from the website of the company, phone calls, live chat, email conversation and recently through social media [P14]. The data acquired through these channels are heterogeneous as characterized below:

- Customer data like Customer ID, Name, Address and Product data like Product ID, Product Name, etc. are structured data.
- An email is a semi-structured data.
- Phone calls (Audio), Product defect photo (Image) are unstructured data.

Example: A customer buys a product X from the Enterprise website. After the product delivery, the customer notices a defect in the product. Now, the customer can either register a complaint in an email with the images of a defective product or else; he can register a complaint to the enterprise customer care through a phone call. The data generated in both of these modes of complaint have to be stored in the enterprise for the effective analysis of the issue claimed by the customer. In this case, all these additional data like emails, pictures of defects received through email from each customer need to be stored and linked to the customer data, which are structured. This enables to later retrieve the information on the customer who registered the complaint.

This huge flow of data has to be effectively stored and handled in-order to attain the objective of CRM such as an increase in profit, revenue and satisfaction of the customer. The Data Vault provides a solution for achieving the goal of a CRM.

3.2 Manufacturing

Manufacturing is the process to convert raw materials or parts into completed goods according to the customer’s specifications or expectations. Usually, manufacturing takes place on a large-scale production line of machinery and skilled workers. Nowadays, automation is important in increasing productivity. Thus, it is essential to define an optimum manufacturing process and to manage it effectively. MES are dynamic information systems that serve the effective execution of manufacturing operations with the help of current and precise data on the plant activities as and when they occur. Product data like Product ID, Product Name, etc. and machine data like Machine ID, Machine Name, etc. from MES are structured. However, the images of the Computer Aided Design (CAD) files, IoT that enables the supervision of the production process [CGP16] and sensor data like temperature, pressure, image, etc. from the machines in the production plant are unstructured.

Example: In a production plant, Machine M is designated to manufacture the product X. To ensure that the machine works properly, the associated sensors to the machine has to be properly monitored. After the product X is manufactured, the finished product has to be then compared with its linked CAD file to ensure that the product manufactured is same as desired. The data generated from sensors like video, image, etc. has to be linked to the machine data, which is structured. For instance, this helps us to identify the machines that malfunctions.

A large-scale industry would comprise of many such machines, which will produce a huge amount of heterogeneous data, which has to be effectively handled to increase the productivity, quality of the product and to reduce the cost of manufacturing.

3.3 Autonomous Car Testing

Autonomous cars are otherwise known as self-driving cars, which sense its environment and drive with less or no human input. In order to perceive their surroundings, a combination of various sensors [Tha17] such as Light Detection and Ranging (LIDAR), Radio Detection and Ranging (RADAR), Camera, Global Position System (GPS) and Inertial Measurement Units (IMU) are used. The data coming from each sensor has to be rightly interpreted to command the car to navigate in the right path and to instruct the car to maneuver or brake when an obstacle is detected. Due to this high complexity, vast amounts of data are generated every 1 km of the drive. According to Intel CEO Brian Krzanich[1], for every 8 hours of driving, every autonomous car will consume and generate approximately 40 Terabytes. Some of the heterogeneous data received during the testing activity are listed below:

- Car data like Car ID, Car Name, etc. and test drive data like test-drive ID, location, climate, etc. are structured.
- Test drive video data, camera sensor data, LIDAR data, etc. are unstructured.

Example: An autonomous car A is subjected to drive test for X hours on a highway. As mentioned earlier, the sensor in the autonomous car produces a tremendous amount of data per second. Along with these data from sensors, let us assume that there is also a video camera to monitor the performance of the car during the test drive, which has to be as well saved for later analysis. Here, the information related to the vehicle and the test case would be structured data, but whereas, the data from sensors and video files are unstructured data. These heterogeneous data have to be rightly linked for us to retrieve any information related to the performance of cars of a particular series. Thus, it is essential to handle these heterogeneous data properly.

1 https://www.networkworld.com/article/3147892/internet/one-autonomous-car-will-use-4000-gb-of-dataday.html

4 Data Vault Modeling

In this chapter, we will discuss the Data Vault modeling in detail to gain a better understanding of the basics. Data Vault is used to model the Enterprise Data Warehouse (EDW), which provides the aggregated, summarized, and consolidated data [LO16]. Data Vault is defined as “a detail-oriented, historical tracking and uniquely Linked set of normalized tables (Hub, Link, and Satellite) that support one or more functional areas of business” [LO16].

Data Vault (DV) is a hybrid combination of star schema (dimensional modeling) and 3NF; these are the two traditional techniques for modeling the data warehouse [KR13]. Listed below are the advantages of the Data Vault modeling over traditional data warehouse modeling approaches [LO16]:

- It provides full agility and accessibility to data in one place.
- Reengineering, which is considered as a massive problem in the data warehouse is avoided.
- It helps to get the data quickly from different data sources and makes it accessible for end users at the earliest possibility.
- It is flexible and provides high scalability.
- It also enables consistency of data and also provides fault tolerance.
- It can be completely automatable.

Data Vault consists of three basic core entities as shown in Figure 4.1. **Hubs** consist of business keys to represent the business objects. **Links** denotes the relationship between two or more Hubs. **Satellites** contain the descriptive information, and it is connected to a single Hub and/or Link.

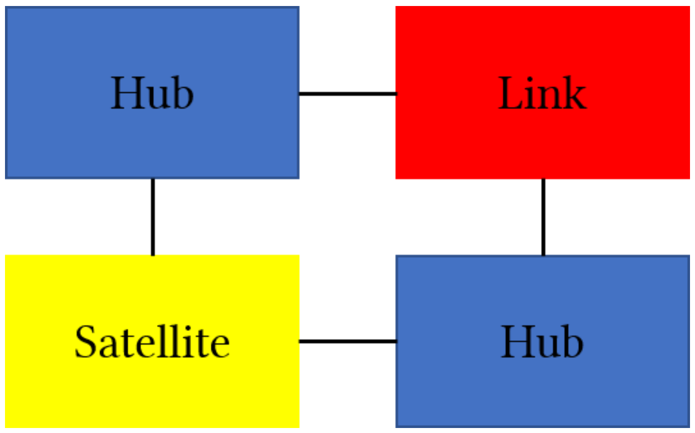


Figure 4.1: Data Vault Entities

Next, we will see in detail about the Data Vault entities in Subsection 4.1, Data Vault architecture in Subsection 4.2 and the difference between Data Vault (DV) 1.0 and Data Vault (DV) 2.0 in Subsection 4.3.

4.1 Data Vault Entities

Data Vault model is made up of three basic entities such as Hubs, Links, and Satellites. The contents of this Subsection are based on various Data Vault references [F14] [H12] [LO16].

4.1.1 Hubs

Hub entities are defined as a collection of unique business keys like *customer_id*, *product_id*, etc. to represent the core business objects such as *customer*, *product*, etc. Whenever a new instance of the business key is introduced to the EDW, a new Hub is generated. Business keys are very helpful to integrate, identify and track the business information in the systems. Hubs contain only business keys, and they do not store any descriptive information. Hubs do not possess any foreign key and are considered to be as a parent table for all other entity tables. As shown in Figure 4.2, a Hub table structure has additional metadata elements.

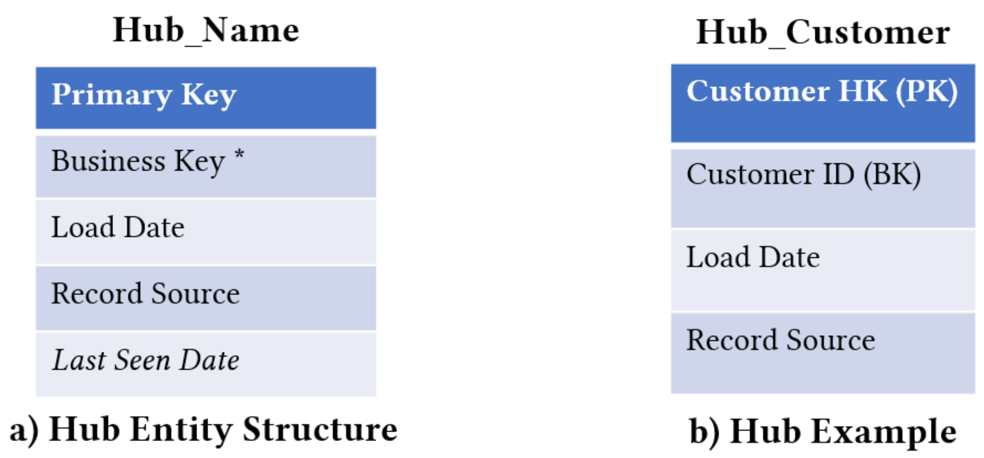


Figure 4.2: Hub Entity Structure and Hub Example

The mandatory elements in a Hub structure are:

- **Primary Key** - In the past, a primary key field represents the sequence number, but now it represents a hash key.
- **Business Keys** - The business key plays a vital role in a Hub because it is the key, which business uses to identify identities. In Figure 4.2, *Customer ID* is the business key (BK) for the *Customer* business object. It is not the sequence number from a source system, but a comprehensible key. In addition to that, the business key can be a *natural key* or a *composite key.*
    - Natural key is the business key if it is unique and provides meaning on its own without adding additional information i.e., email address, social security number are some of the examples for a natural key.
    - If the business key of an entity is composed of more than one data field, then it is called a composite key. A composite key is also known as a smart key or intelligent key. For example, Vehicle Identification Number (VIN) combine different sections like a World Manufacturer Identifier (WMI) code, a Vehicle Descriptor Section (VDS) and a Vehicle Identifier Section (VIS) [LO16].

- **Load Date** - This field describes when the business key first arrived in the data warehouse. This load date is a system generated field in the data warehouse. It indicates a constant timestamp view of the data as it appears in the data warehouse and this should never be altered.
- **Record Source** - It identifies the original source system of the business key. This is a key attribute in maintaining the auditability in the data warehouse.

The optional element in a Hub structure is:

- **Last Seen Date** - It represents when the business key was seen for the last time in the source system. This field gets updated when the business key is seen in the source system. It can be used to identify when the business key is deleted from the source system. For example, business keys with a last seen date of more than a year ago can be deleted from the Data Vault.

4.1.2 Links

A Link represents the relationship or association between two or more business objects (Hubs) or the same Hub twice. A Link entry is established for the first time when a new unique relationship appears in the EDW. A Link contains primary key and foreign keys, but it does not have any descriptive data. In addition to that, Links make the Data Vault model more flexible i.e., to add new functionality to the model; it would only require to add new Hubs to the model and connect them using Links to the existing Hubs in the model. As shown in Figure 4.3, a relationship known as *Link_Order* exists between two Hubs namely *Hub_Customer* and *Hub_Product*.

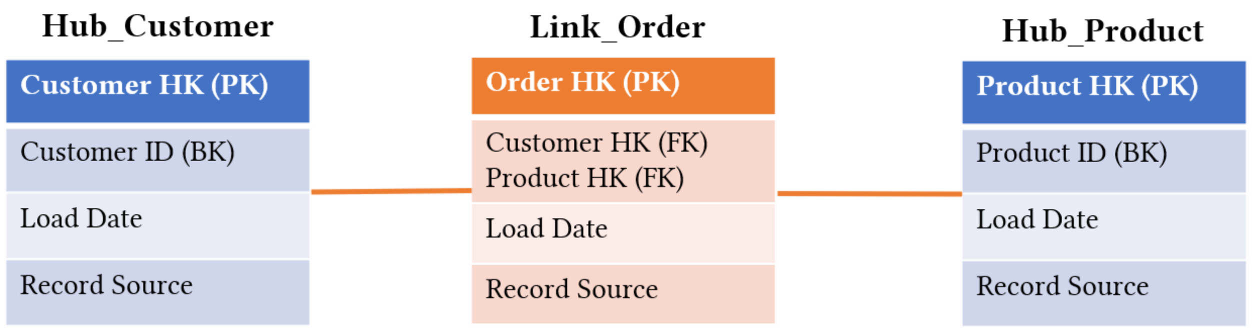


Figure 4.3: Example for a Link Entity

Similar to the Data Vault Hub, a Link table structure has additional metadata elements as shown in Figure 4.4.

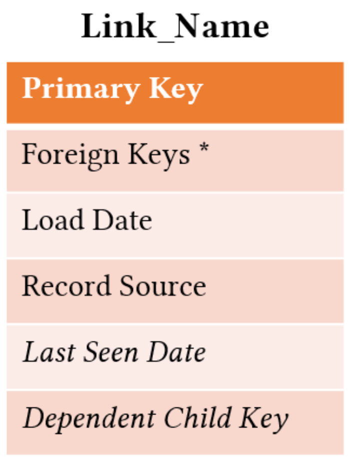


Figure 4.4: Link Entity Structure

Many-to-Many Relationship

Even though relationships can be one-to-one, one-to-many or many-to-many, the Data Vault Links can be expressed only as a many-to-many relationship in order to increase flexibility, scalability, etc. Next, we will see in detail how many-to-many relationship provides flexibility and scalability.

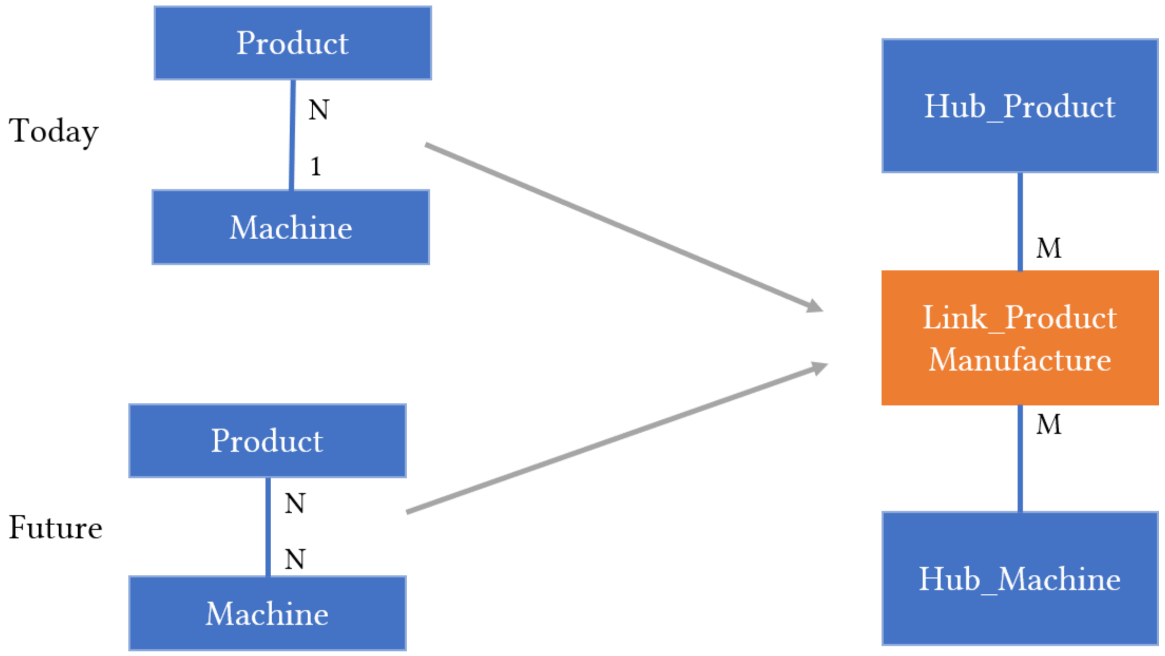


Figure 4.5: Link Modeled as a Many-to-Many Relationship to Provide Flexibility [GL11]

a) Flexibility

To explain how a Link supports flexibility, a scenario is considered (see Figure 4.5 on the left), where a business rule is defined today as: “one machine can produce many products, but each product must be produced by only one machine”, and the system supports only this business rule. Everything runs smoothly until in future it is decided to modify a business rule as: “2 or 3 machines can produce more products”. In this case, data warehouses face a problem because the structure remains rigid when modeled as a one-to-one relationship [LO16]. To adapt to the new business rules, the data warehouse has to be re-engineered. To provide a solution to this problem in the Data Vault, a Link is introduced and modeled as a many-to-many relationship (see Figure 4.5 on the right). The Link entities, which are expressed as a many-to-many relationship, help the physical design to handle the changes in a business rule and data, in such a way that the existing datasets or processes remain unaffected. Therefore, with the help of a Link entity with regards to cardinalities, the change of business rules in the present, past and future can be handled without re-engineering.

b) Scalability

Links provide scalability in the Data Vault model and to understand it, we will take the reference of the model as shown in Figure 4.3, to extend this small model into a larger model by adding more Hubs and Links based on the use case. As shown in Figure 4.6, a new Hub known as *Hub_Supplier* can be added to the existing Hub *Hub_Product* with the help of a new Link *Link_Supply* without modifying the existing model. Thus, scalability is achieved.

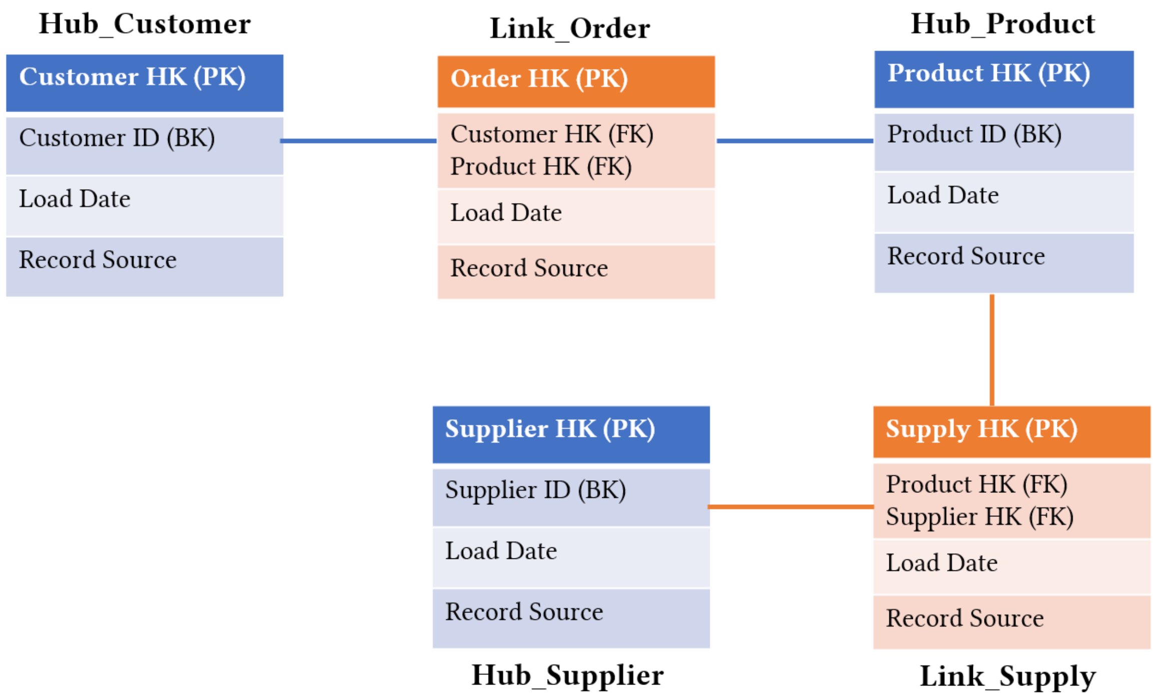


Figure 4.6: Example to Show that a Link Provides Scalability

The mandatory elements in a Link structure are:

- **Primary Key** - It is similar to a primary key field in a Hub entity structure. A hash key for a Link is generated from the referenced business key i.e., from the business keys of the Hubs a link is connected. Considering that the *Link_Order* refers to two Hub tables, the hash keys of these two Hub tables are stored as foreign keys in a Link table. The hash key *Order HK* is then generated from the referenced business key.

- **Foreign Keys** - This field contains the hash keys of the Hubs to which a Link is connected. Usually, it contains two or more foreign keys because, in most of the cases, a Link is connected to two or more Hubs. Even if a Link is connected to only one Hub, it does contain at least two foreign keys. For example, in the social network, User A can be a friend of User B. In Figure 4.7, *Facebook User* is considered to be a Hub and to show the relationship between Facebook users, *Link_Friends* is established twice between the *Hub_Facebook User*. The *Link_Friends* possess two foreign keys *User HK*.

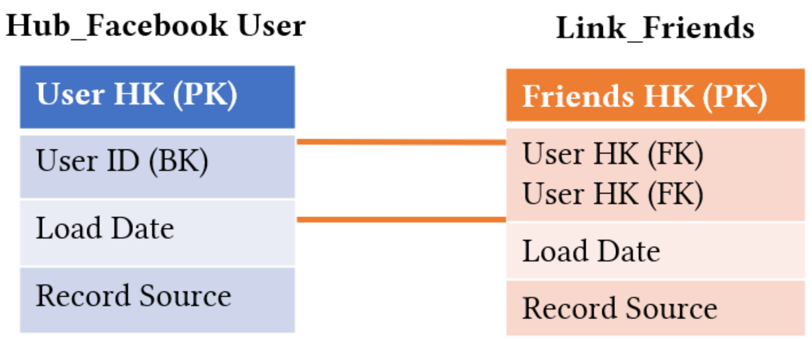


Figure 4.7: Example for a Link that is Connected to Only One Hub

- **Load Date** - It is similar to the load date in a Hub entity structure to identify when the data/association was first loaded.
- **Record Source** - It is similar to the record source in a Hub entity structure to identify from where the data are coming.

The optional elements in a Link structure are:

- **Last Seen Date** - It is similar to the last seen date in a Hub entity structure.
- **Dependent Child Key** - This field gives meaning when it is combined with other key information like business keys. Dependent child key in link structure is used as an identifying element. To highlight the need of a dependent child key, an extreme case scenario is considered where the same product occurs twice in a sale due to different discounts. In such case, the dependent child key is used in a Link to differentiate the two products from each other. If a dependent child key is present in a Link structure, then a hash key in a Link is generated from that dependent child key and the referenced hub business keys. For example, in Figure 4.8 below, *Product* and *Invoice* as Hubs and the relationship between these two hubs as *Invoice Order*. Here, *Line Item Number* is a dependent child key since it is dependent on the business key *Product ID*. Thus, a hash key Order HK for a Link is generated from *Invoice No*, *Product ID* and *Line Item Number*.

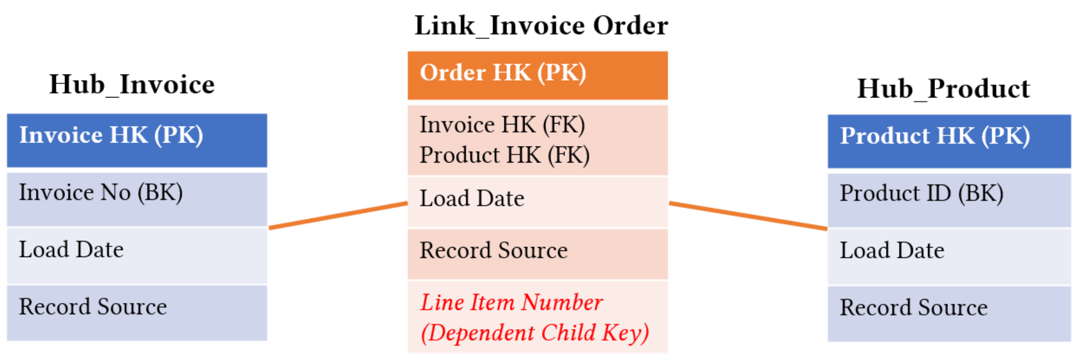


Figure 4.8: Example for a Dependent Child Key

Order HK	Invoice HK	Product HK	Line Item Number	Load Date	Record Source
1fhjd23hjs...	ag3883nxcjk..	uwi2378nm..	1	2010/10/10	Product
1fhjd23hjs...	ag3883nxcjk..	uwi2378nm..	2	2010/10/10	Product

Table 4.1: Link Table with Dependent Child Key

4.1.3 Satellites

Satellites are used to store attributes/descriptive data of Hubs or Links. The primary purpose of a Satellite is to track the changes in the system by capturing all the changes happening in the descriptive data. If the data in the source system changes, a new entry is added as we need to preserve the history of the data using a Satellite. A Satellite can be attached to either a Hub or a Link, but on any case, it should have only one parent table. Furthermore, a parent table can have one or more Satellites, but a Satellite cannot be a parent to any other table, and due to this reason, they don't generate hash keys.

Similar to a Hub and a Link in the Data Vault, a Satellite table structure has additional metadata elements as shown as in Figure 4.9.

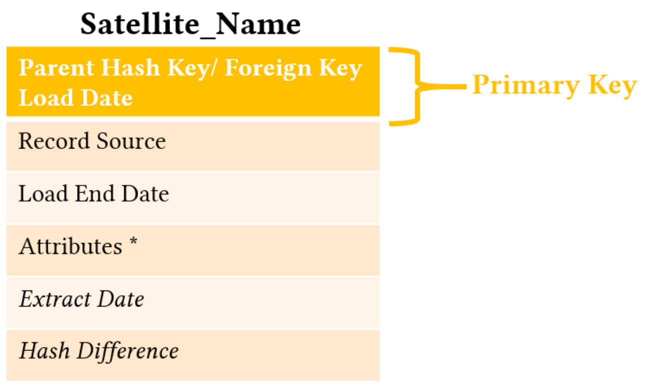


Figure 4.9: Satellite Entity Structure

Splitting Satellites

It is possible to store all the descriptive data of a business object in the attributes of one satellite, but this is not our aim. As an alternative, it is suggested to split the data among different satellites. It is recommended to split the raw data by source system initially and then if the tables are big, we can further split it by frequency of change. For example, the *Product* attributes are added to a single satellite table as shown in Figure 4.10. Here, the attributes of *Product* like *Name*, and *Description* do not often change, whereas the attributes like *Quantity*, and *Price* change often. If we store all the attributes in one Satellite then in-case if there exists an attribute that changes frequently, a new record will be produced each time when the attribute changes. This new record would as well consist of non-changing attributes in a Satellite to track the changes as shown in Table 4.2 (b). This results in consumption of additional memory and this problem can be solved by splitting the data by the rate of change and adding the attributes that changes often to one Satellite and the attributes that do not often change to another Satellite.

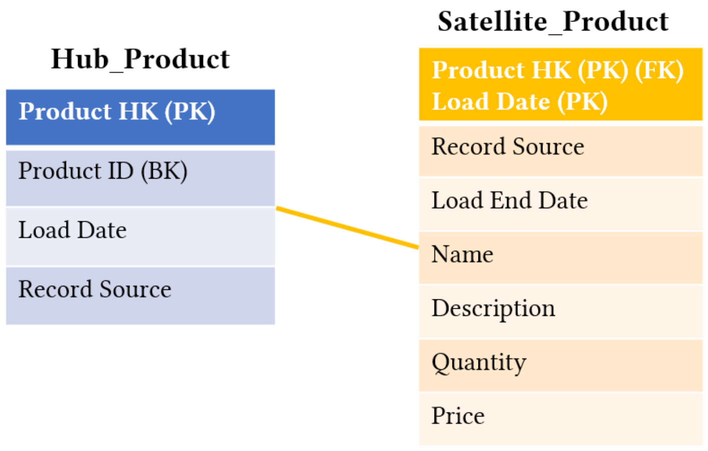


Figure 4.10: Example for Satellite (Split Based on Source System)

a) Satellite data split by Source- Before Update

Product HK	Load Date	Load End Date	Record Source	Name	Description	Quantity	Price
Fhsjs3445....	2016-03-17 10:12:22	9999-12-31 24:00:00	Product.xml	Bag	Leather Bag	10	300

b) Satellite data split by Source- After Update

Product HK	Load Date	Load End Date	Record Source	Name	Description	Quantity	Price
Fhsjs3445....	2016-03-17 10:12:22	2016-04-17 10:12:21	Product.xml	Bag	Leather Bag	10	300
Fhsjs3445....	2016-04-17 10:12:22	2016-05-17 10:12:21	Product.xml	Bag	Leather Bag	15	500
Fhsjs3445....	2016-05-17 10:12:22	9999-12-31 24:00:00	Product.xml	Bag	Leather Bag	12	370

Table 4.2: Satellite Data with Change in Frequency are Split by Source System

In Figure 4.11, it can be seen that Satellites are split based on the frequency of change in *product* attributes, i.e. *Product* attributes like *Name*, and *Description* that does not change often are added in one Satellite and the product attributes like *Quantity*, and *Price* that often changes into another Satellite. Thus, whenever new records are produced for *Quantity* and *Price*, only that particular table is updated. With this approach, the data replication of column data can be avoided and also the row size can also be reduced as shown in Table 4.3 (b).

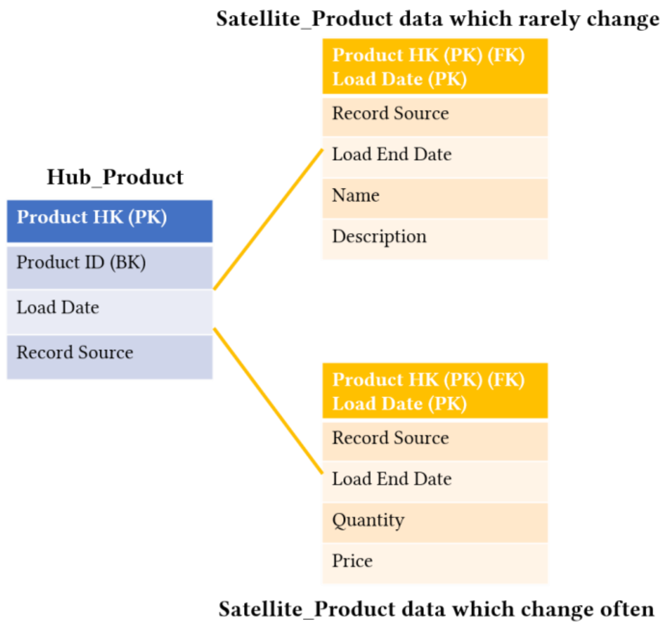


Figure 4.11: Example for Satellite (Split Based on Rate of Change)

a) Satellite data split by Rate of Change- Before Update

Product HK	Load Date	Load End Date	Record Source	Name	Description
Fhsjs3445....	2016-03-17 10:12:22	9999-12-31 24:00:00	Product.xml	Bag	Leather Bag

Product HK	Load Date	Load End Date	Record Source	Quantity	Price
Fhsjs3445....	2016-03-17 10:12:22	9999-12-31 24:00:00	Product.xml	10	300

b) Satellite data split by Rate of Change - After Update

Product HK	Load Date	Load End Date	Record Source	Quantity	Price
Fhsjs3445....	2016-03-17 10:12:22	2016-04-17 10:12:21	Product.xml	10	300
Fhsjs3445....	2016-04-17 10:12:22	2016-05-17 10:12:21	Product.xml	15	500
Fhsjs3445....	2016-05-17 10:12:22	9999-12-31 24:00:00	Product.xml	12	370

Table 4.3: Satellite Data Split by Rate of Change

The mandatory elements in a Satellite structure are:

- **Foreign Key** - This field contains the hash key of the parent table. Here, a parent table can be a Hub or a Link depending on to which a Satellite is attached to.
- **Load Date** - It is similar to the load date in a Hub and a Link entity structure. In a Satellite structure, load date and parent hash key together form a primary key. A parent hash key and load date will be helpful in tracking the historical data.
- **Record Source** - It is similar to the load date in a Hub and a Link entity structure.
- **Load End Date** - This field is used to represent the date and time after which the entries in a Satellite are not valid. When the new entry has a current load date, the last satellite entry that was valid just before the loading of the new entry is updated to reflect the new load end data. Load end date is the only field, which can be updated in a Satellite entity. For example in Table 4.2 (b) it can be seen that the Column *Load End Date* has a value that overlaps the column Load Date. This field is used in a Satellite to improve the performance while extracting the data from Data Vault.
- Attributes - These fields store the descriptive data of a Hub or a Link.

The optional elements in a Satellite structure are:

- **Extract Date** - It represents the date and time when the attribute was extracted from the source system.
- **Hash Difference** - It represents the hash value generated for all the attributes in the Satellites. This field will be beneficial to identify the changes (if any) in the attributes easily and add the new attributes when there is a change in a Satellite. In this way, we can avoid comparing each attribute separately to identify whether there was any change or not.

4.1.4 Rules for Modeling the Data Vault Entities

There are some sets of rules for modeling Hubs, Links and Satellites, which need to be taken into consideration while designing the Data Vault model. Some of the important rules listed below are based on various references [F14] [L18] [LO16]:

Hub Rules

- It is necessary for a Hub to have at least one unique business key.
- Direct Hub-to-Hub relationship is not allowed. Hubs should be connected only through a Link.
- A Hub should have at least one Satellite. A Hub without a Satellite does not give any meaning, and this practice should be avoided.
- A Hub's load date should be a metadata element in a Hub; it should never be the part of a Hub's primary key.
- A Hub's record source cannot be a part of a Hub's primary key.

Link Rules

- A Link table solves recursive relationship problems.
- Satellites are optional for a Link structure.

- A Link's load date should be a metadata element in a Link; it should never be the part of a Link's primary key.
- A Link can be defined as a relationship, hierarchy or transaction.

Satellite Rules

- A Satellite cannot be a parent table to any other table.
- A Satellite cannot have its own key. It has a parent table key as a foreign key, which is also a part of primary key.
- A Satellite's load date is a part of Satellite's primary key.
- A Satellite's main purpose is to store the historical data. A Satellite can be split or combined at any time without changing or losing the historical data.

4.2 Data Vault 2.0 Architecture

Two well-known ways to build the data warehouse are Kimball and Inmon's approach [KR13]. Kimball's approach is based on a two-layer architecture consisting of the staging area and the data warehouse area. Inmon's approach is based on a three-layer architecture consisting of the staging area, the data warehouse area, and the data access area. These approaches are not optimal concerning extensibility, and they lack as well in various dimensions of scalability like workload, data complexity, query complexity, availability. The Data Vault 2.0 architecture provides a solution to these problems by modifying the Inmon's data warehouse architecture, which will be discussed in detail in this chapter. The Data Vault 2.0 architecture is designed in such a way to support the Relational Database Management System (RDBMS) and NoSQL (Not Only SQL).

The goal of this architecture is to move the business rules, which are used between the source and the data warehouse towards the end user. The advantage of having this type of structure is to ensure quick adaption to changes. The Data Vault 2.0 architecture comprises three layers: the staging layer, the enterprise data warehouse layer, and the information delivery layer as shown in Figure 4.12 [LO16].

- **Staging layer** - The structured data from various sources are fed into the staging area. To load the batch data into the data warehouse, staging layer is used. In the staging area; the data that are collected from different sources are kept in the raw (original) format, and no historical information is stored. The primary purpose of the staging area is to reduce the load on the source system. *Hard business rules* are applied to the incoming data, which is extracted from the source system and loaded into the staging area. Hard business rules do not change the meaning of the data but only change the way on how the data are stored. Let us assume that the data received to the target column (INTEGER) from the source system exceeds its predefined size. In this case, the hard business rule will change the data type of INT (integer) to BIGINT. Metadata are important to load the data into the enterprise data warehouse layer, and hence it has to be added in advance to the staging layer. Sequence number, timestamp/load data, record source, hash key/sequence number are the metadata fields, which need to be included.

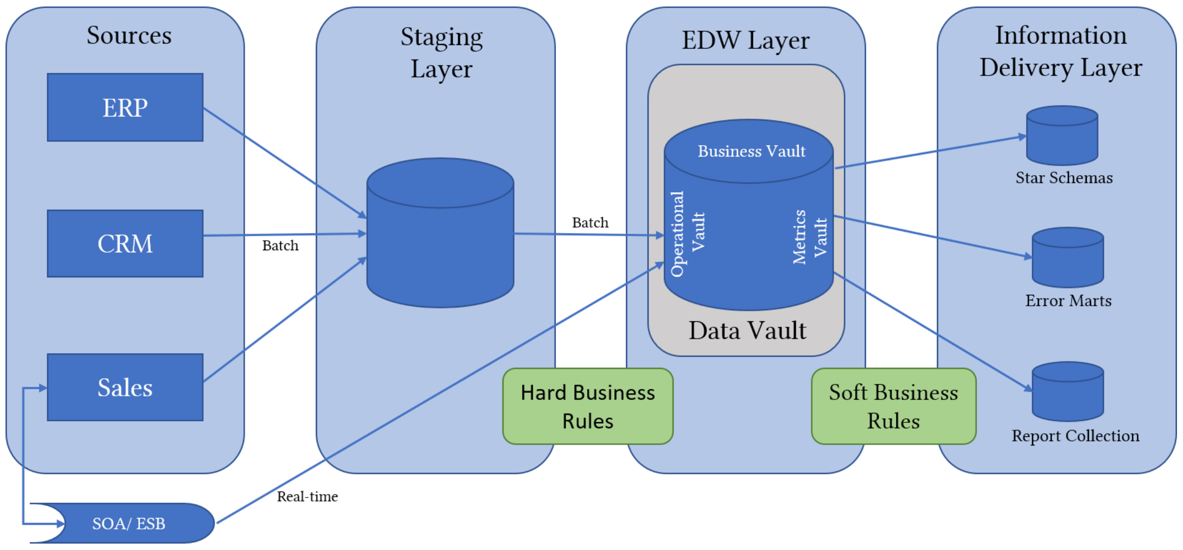
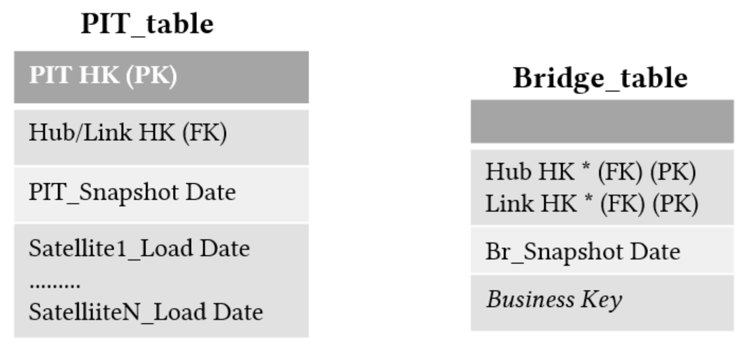


Figure 4.12: Data Vault 2.0 Architecture [LO16]

- **Enterprise data warehouse layer** - The EDW layer is the second layer in the Data Vault 2.0 architecture, which the end users cannot access directly. This layer is also known as Raw Data Vault layer, which is modeled after the Data Vault 2.0 modeling technique [LO16]. Its primary purpose is to hold the historical data at the granular level received from the source system. The data in this layer is nonvolatile, and the changes in the source system are tracked with the help of Data Vault structure using business keys. As said earlier, the batch data are loaded to EDW layer from the staging area, but in case of the real-time data, it is loaded directly into EDW layer with the help of Enterprise Service Bus (ESB). In the Data Vault, there are three components: Operational Vault, Business Vault, and Metric Vault.
    - ***Operational Vault*** is used to store the raw data, which is fed into the data warehouse from the staging layer. Operational Vault is directly accessed by operational systems, and Data Vault tables (Hubs, Links and Satellites) are modeled.
    - ***Business Vault*** is an intermediate layer between the Data Vault (EDW layer) and the information delivery layer where the *soft business rules* are applied. Soft business rules enforce the information requirements of the business as stated by business users. By applying the soft business rules, the meaning of the data changes into useful information for the business. To consolidate the data from different sources (or) aggregation of data into categories based on age group are an example of soft business rules. Business Vault has the *Point in Time (PIT) table* and *Bridge table* and known as query assistance tables, which help to speed up the extraction process from the Data Vault.

      PIT table is a modified version of a Satellite table, and it is created from a single Hub or a Link. A PIT table is introduced in the Data Vault model to improve the performance of a Hub or a Link, whenever its query performance is low. A PIT table stores a hash key of a Hub/Link, load date of all Satellites and the snapshot date of the PIT table. To identify when the specific record was loaded into the PIT

table, the snapshot date attribute is used. Unlike Satellites, PIT table does not store the descriptive attributes. It is used to figure out, which version is valid on a specific date. Here, we consider that a Hub Customer contains multiple Satellites. Each PIT table may have several versions of a particular data with different load dates. In Table 4.4, it can be seen that the valid versions for the Customer hash key *dfdfjk11* on *2010-04-01* are *2009-11-11* for *Sat1*, *2006-02-01* for *Sat2* and *2000-03-01* for *Sat3*.
Bridge table is considered as a special link table, created from multiple Hubs and Links. It stores hash keys of Hubs and Links that are often queried together with the snapshot date. In addition to this, it can also have the business keys.
With the help of PIT table and Bridge table, we can reduce the required joins needed for the query, and this leads to improve the performance of queries on the Raw vault.



a) PIT Table Structure **b) Bridge Table Structure**

Figure 4.13: Structure for PIT Table and Bridge Table

Customer HK	Snapshot Date	Sat1_Load date	Sat2_Load date	Sat3_Load date
dfdfjk11	2010-01-01	1999-09-09	2000-01-01	2000-03-01
dfdfjk11	2010-02-01	1999-09-09	2005-02-01	2000-03-01
dfdfjk11	2010-03-01	2004-03-07	2006-02-01	2000-03-01
dfdfjk11	2010-04-01	2009-11-11	2006-02-01	2000-03-01

Table 4.4: Example for PIT Table

- In ***Metrics Vault***, runtime information is captured and recorded including process metrics, technical metrics and run history such as usage of RAM, load in CPU. It is a component that is very helpful with error inspection, root cause analysis, and performance evaluation. Like the EDW layer, metrics vault is modeled only after modeling the Data Vault 2.0.

- **Information delivery layer** - This is the layer, which end users can access via data marts and from here, they derive the business insights in the way they want. The data in these data marts are aggregated, subject-oriented, flat and highly indexed, completely prepared for a specific use case. The data drawn from data marts using Online Analytical Processing (OLAP) cubes are used for decision-making, prediction, artificial intelligence, and machine learning. Apart from this, the quality of data are ensured, and poor/faulty data are stored in an error mart for further analysis.

In Data Vault 2.0 architecture, the staging layer, the EDW Layer and the Information delivery layer are mandatory, and the optional extensions are the Operational Vault, the Business Vault and the Metric Vault.

4.3 Data Vault 1.0 vs. Data Vault 2.0

Data Vault 1.0 was introduced by Linstedt in 1990, it is very much focused on the Data Vault modeling i.e. a commitment to the logical and physical data models that build the raw enterprise data warehouse whereas Data Vault 2.0 was introduced by Linstedt in 2013 as an extension that possesses several of the necessary components for the achievement in the field of business intelligence and data warehousing [LO16]:

- DV 2.0 Modeling: Changes to the model to interact with NoSQL and Big data systems to improve performance and scalability.
- DV 2.0 Methodology: Following Agile and Scrum best practices.
- DV 2.0 Architecture: Handles unstructured data and big data integration using NoSQL and big data systems.
- DV 2.0 Implementation: Focuses on pattern based, automation, generation, and Capability Maturity Model Integration (CMMI) level 5.

The main difference between Data Vault 1.0 and Data Vault 2.0 is the replacement of sequence number with hash key [L14]. It is important to know why this change was introduced. In Data Vault 1.0, sequence number is used as a primary key to identify the business objects. The sequence number has various limitations like [LO16]:

- In Data Vault 1.0, a primary key field represents the sequence number. A sequence number is a unique number assigned to recognize the order of records in the source system. These numbers are generated in the data warehouse, which is used to identify and refer the records in other tables in the traditional data warehouse. These sequence numbers cause dependencies in the loading process because all related dependencies must be loaded first before loading the destination. These dependencies will impact the performance of the loading process by making it slow. For example, in order to load a table, which stores the address of the customer, the customer table has to be loaded first. Thus, being able to use the sequence number to identify a particular customer. Data Vault 2.0 overcomes these dependencies by substituting the sequence number with a hash key. This hash key is generated from the business key using hash algorithms such as Secure Hash Algorithm-1 (SHA-1), Message-Digest Algorithm (MD5).

- Sequence numbers are generated using sequence generator. These numbers need to be synchronized to avoid any duplication. This synchronization can be difficult in big data environments as data are coming at high volume and velocity.
- The sequence number must be same as before when restoring a parent table. If not, the references in Link and Satellite will either be invalid or wrong, i.e. if the parent table gets deleted and when later retrieved, it should possess the same sequence number as before.
- It is easy to use sequence numbers but they are limited by its space and this causes scalability issues.
- The sequence number must be avoided for data distribution and partitioning in Massively Parallel Processing (MPP) for the reason that the queries can result in hot spots in MPP environments when obtaining the data out of it i.e., one area is accessed multiple times within short duration. MPP is nothing but to process many operations in parallel, which are done by many processing units at the same time.

To overcome these limitations, hash keys were introduced in the Data Vault 2.0, and its advantages over sequence numbers are:

- Since hash keys are autonomously calculated in the process of loading, it is not needed to have a lookup into other Data Vault entities. In general, Input/Output (I/O) performance is needed for a lookup into another table to obtain the sequence number for a certain business key. However, only CPU performance is required to compute a hash key, which is frequently preferred over I/O performance since it results in better resource consumption and better parallelization.
- Hash functions are used to generate hash keys, which ensures that the same hash key gets produced for the same input.

The reason why dependencies are avoided in the Data Vault 2.0 is that the foreign keys for the links and satellites that refer to the hubs can be calculated beforehand, meaning that a Hub entity does not have to be inserted to know its key as is the case with sequence numbers as shown in Figure 4.14 and Figure 4.15. Instead, when knowing the business key of a Hub entity, a hash key can be calculated and added to a Link/Satellite. Likewise, with respect to Link entities, when the hash keys for the Hubs to which a Link connects are known, the key of a Link can be calculated and a Satellite can refer to this key. A hash key allows the EDW to be used in multiple environments like on-premise databases, Hadoop and in Cloud storage environment because those don’t support sequence numbers [LO16].

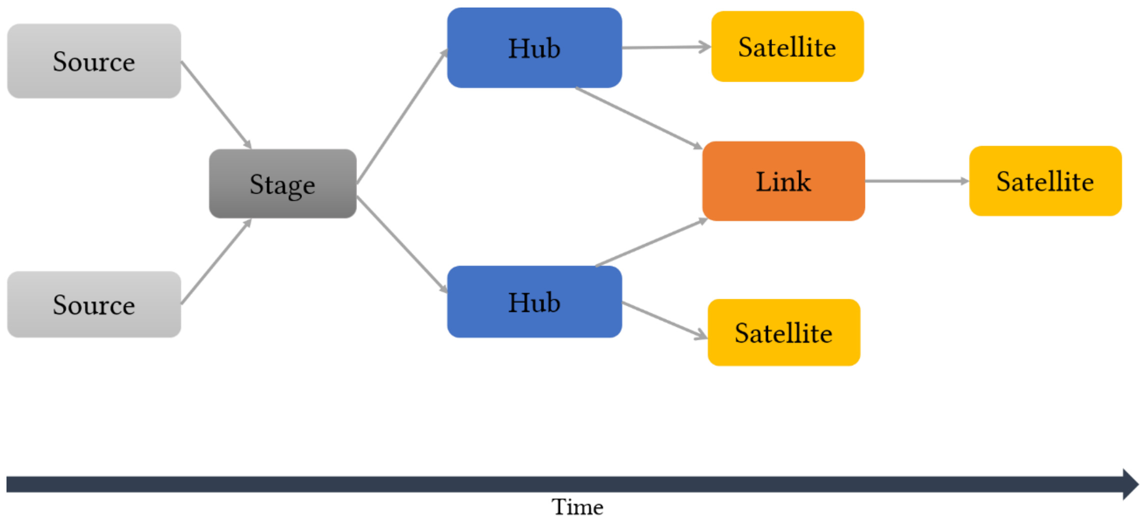


Figure 4.14: Data Vault 1.0 Loading Process with the Sequence Number [L016]

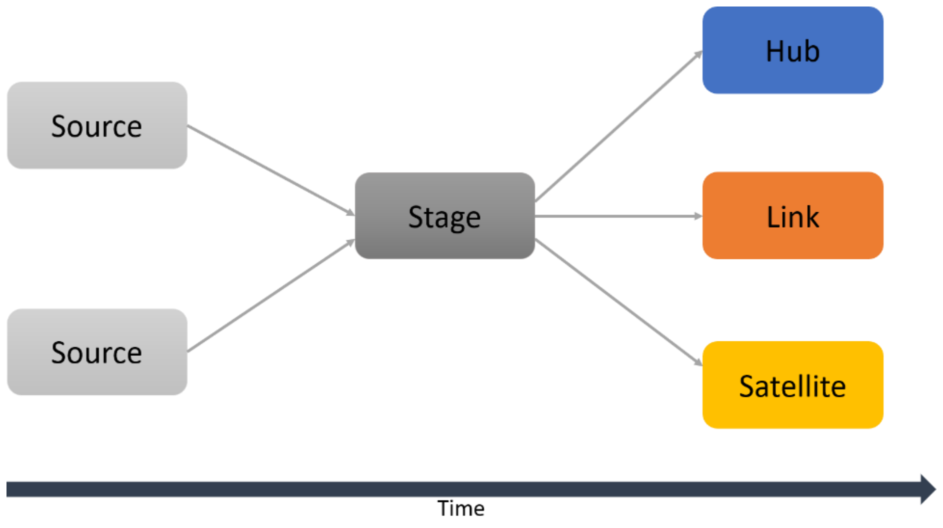


Figure 4.15: Data Vault 2.0 Loading Process with a Hash Key [L016]

5 Existing Approaches to Integrate Semi-Structured Data into the Data Vault Model

Till now we have seen the fundamentals of the Data Vault and how to model the structured data in the Data Vault. However, the presented use cases have semi-structured and unstructured data as well. Therefore, in this chapter, we will look into some of the existing approaches to integrate semi-structured data into the Data Vault model. It should be observed that there are some approaches to integrate semi-structured data in the Data Vault, but it does not cover all aspects of semi-structured data. Also, there are no existing approaches to express how to integrate unstructured data. Here, we will discuss how to transform JSON data to Data Vault in Subsection 5.1 and the process for creating an XML schema represented as the Data Vault model in Subsection 5.2.

5.1 Translation Rules for JSON Data into the Data Vault

To integrate the data from various data sources into the Data Vault model and to keep track of changes in relational and NoSQL systems, Cernjeka et al. have developed a metamodel to translate a NoSQL document store that uses *JSON* data as shown in Figure 5.1 into the Data Vault model [CJJ18]. In general, a metamodel is a model by itself that describes only about the structure but not about the content of another model. NoSQL document stores are based on the key-value concept, and the data are stored in a document structure. JSON and Binary JSON (BSON) are the formats used to represent the data in the document stores. Some of the document stores are MongoDB, CouchDB, etc. The focus of the paper [CJJ18] mainly lies on Mongo DB.

The main entities of MongoDB are collection, document, field, and database. It is essential to understand the terminologies used in MongoDB in relation to the relational database terminologies, which are described below [CJJ18]:

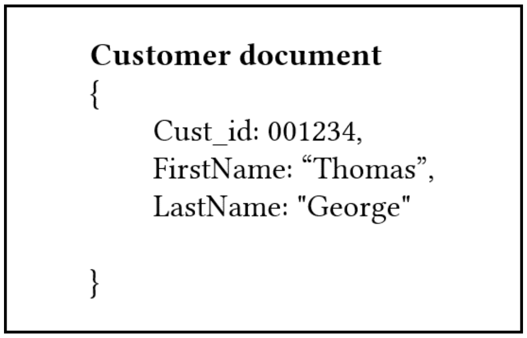
- The term ‘table’, which consists of row and column in the relational database represents ‘*collection*’ in MongoDB.
- The term ‘*record*’, which represents a row in the relational database and ‘*document*’ in MongoDB.
- The term ‘*document*’ represents the ‘*field and value*’ data structure. Document ‘*field*’ represents the ‘*column*’ and document ‘*value*’ denotes the intersection of row and column of ‘*actual data*’ in a relational model.

‘*Field and value*’ in document store are very similar to the term ‘*key and value*’ pairs used in key-value stores. In the key-value store, ‘*key*’ represents the unique field, which will be helpful to identify the ‘*value*’. Likewise, in the document store, ‘*field*’ represents the unique id like document_id and ‘*value*’ can be documents, arrays or array of documents.

Due to the complex structure of the data, instead of representing the data in tables it is expressed as JSON documents. JSON documents[2] are easy to read and write because it is a language independent and it is written in text format. JSON document has the following structure as shown in Figure 5.1.

2 https://www.json.org/index.html

- Each JSON document is considered to be as an object, which is enclosed between '{' and '}'. Key-Value pairs are represented between these curly brackets.
- Each key/field is separated from the value by ':' and if there are subsequent values they are separated by ",". For example, *FirstName* is the Key and *Thomas* is the value for that key.



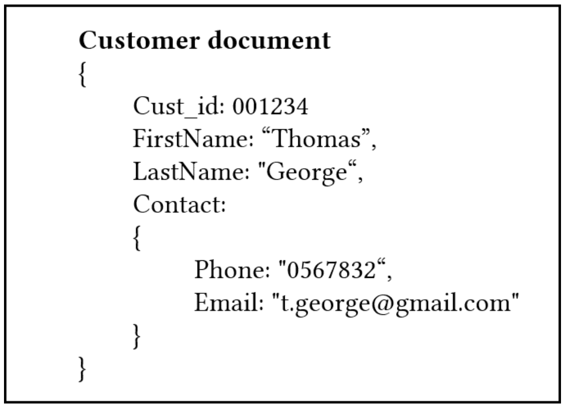
```
Customer document
{
        Cust_id: 001234,
        FirstName: "Thomas",
        LastName: "George"

}
```

Figure 5.1: JSON Structure

The relationship explains how the data in JSON documents are connected to each other. The following are the two ways to represent JSON document relationships [CJJ18]:

- *Embedded documents* show the relationship between the data in a single JSON document structure and this kind of relationship can be seen in denormalized models. They represent one-to-one, one-to-many and many-to-one relationships. The embedded document requires fewer reads but more writes, and also it redundantly stores data to reduce the number of documents to be accessed. For example, in Figure 5.2, *Contact* details are embedded within the *Customer document*.
- *Reference documents* are also known as normalized data models, which show the relationship between data by using links or references between the documents. They represent many-to-many relationships, and when compared to embedded documents they are more flexible because it requires more reads and less writes. For example, in Figure 5.3, *Customer document* is referenced by *Contact document* with the help of *Customer_id*.



```
Customer document
{
        Cust_id: 001234
        FirstName: "Thomas",
        LastName: "George",
        Contact:
        {
                Phone: "0567832",
                Email: "t.george@gmail.com"
        }
}
```

Figure 5.2: Embedded JSON Document

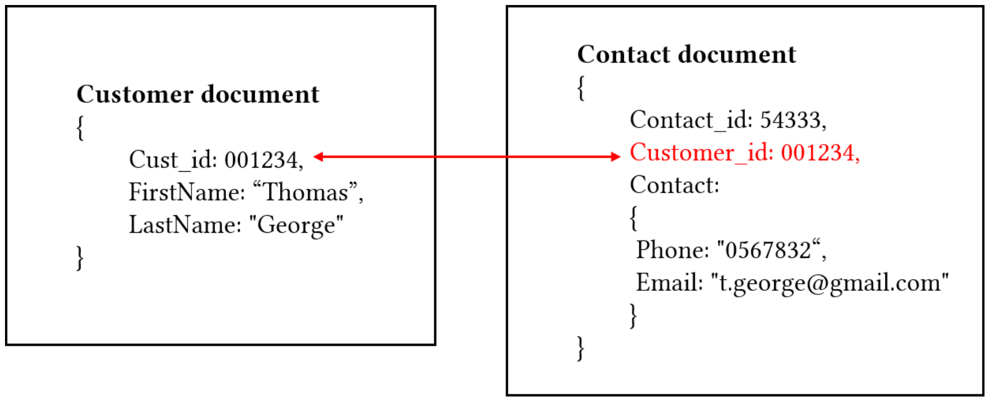


Figure 5.3: Reference JSON Document

In the paper [CJJ18], a set of translation rules has been defined to transform JSON data into the Data Vault. These translation rules were used in the integration process.

Translation Rule 0 (TR0): MongoDB document database has a collection of documents that are represented as a JSON object. Hence, collections are the starting point, which needs to be translated into the Data Vault entities such as Hubs, Links and Satellites [CJJ18].

According to TR0, the MongoDB document represented as a JSON object for example *Customer document* consists of various field-value pairs like *FirstName* as field and *Thomas* as value as shown in the above Figure 5.1.

Translation Rule 1 (TR1): Each document_id in JSON document is translated into the business key, while a hash key is generated from the business key and added to a Hub [CJJ18].

According to TR1, document_id is created for every document, and it helps to identify each document uniquely. Therefore, this document_id (*Cust_id*) is translated into Hub_id (business key) in the Data Vault model. Based on this rule, a Hub *Customer* is created with *cust_id* as the business key, and with the help of the business key, a hash key *Customer HK* is generated as shown in Figure 5.4.

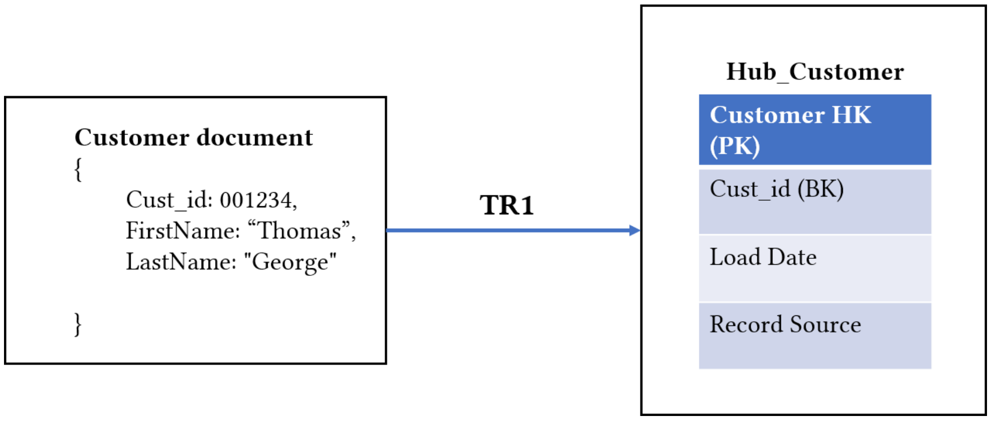


Figure 5.4: For JSON Data - Hub Entity is Created According to TR1

Translation Rule 2 (TR2): The other non-id fields of the documents are transformed as attributes of a Satellite entity [CJJ18].

According to TR2, the non-id fields like *FirstName* and *LastName* in the document are represented as attributes in a satellite entity as shown in below Figure 5.5.

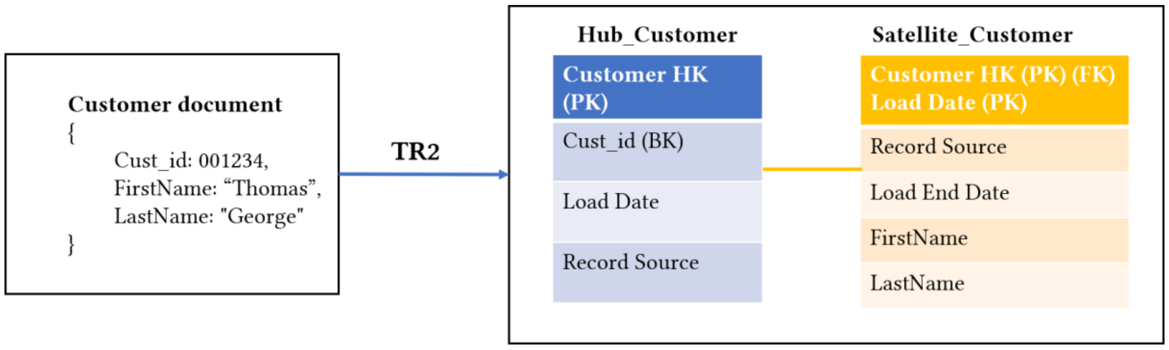


Figure 5.5: For JSON Data - Satellite Entity is Created According to TR2

Translation Rule 3 (TR3): A document reference to another document is transformed into a Data Vault Link entity relating the current Hub (document) and the parent Hub (referenced document) [CJJ18].

Applying TR3 to Figure 5.3, the parent document (*Customer document*) and the current document (*Contact document*) are created as Hubs, and they are connected with the help of reference value (*Customer_id*) in the current document, which acts as a data vault link between both Hubs. As shown in Figure 5.6, the current Hub (*Hub_Contact*) is connected through Link entity (*Link_Customer Contact*) with the parent Hub (*Hub_Customer*).

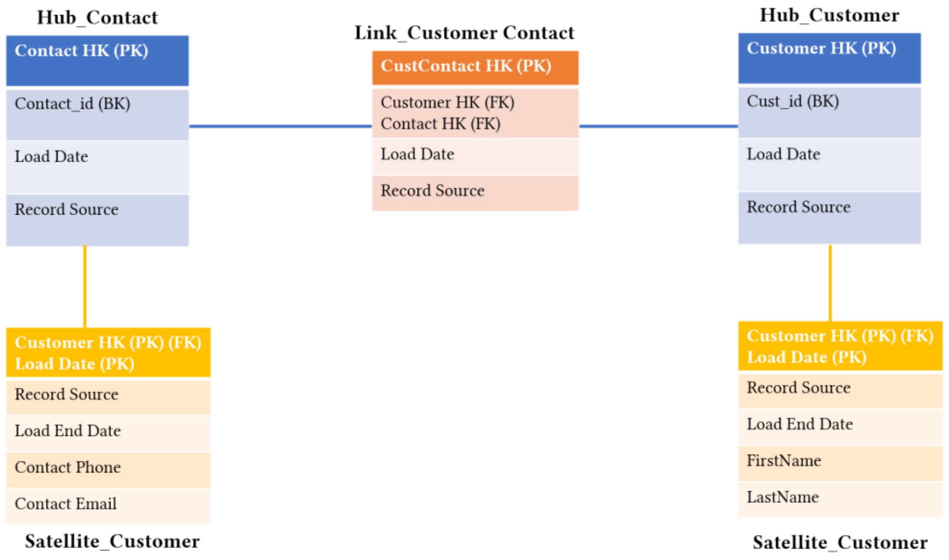


Figure 5.6: For JSON Data - Link Entity is Created According to TR3

Translation Rule 4 (TR4): An embedded document is transformed into a Data Vault Satellite entity, which is connected to the current Hub (parent document). In that circumstance, TR1 To TR3 are applied accordingly [CJJ18].

As shown in Figure 5.2, if *Contact* details like *Phone* and *Email* (non-key id fields) are embedded within the parent document (*Customer document*) then according to TR2, that embedded document is translated into a Data Vault Satellite. All the descriptive attributes of a business object (*Hub_Customer*) are stored in one Satellite. Later, after splitting a Satellite by source system as shown in Figure 5.7 (a), a Satellite can be split based on the rate of change, which is not a part of TR4. Let us assume, *Name* attributes won't change frequently, and attributes of *Contact* details may change often, so we have split a Satellite into two Satellites as *Satellite_Customer* and *Satellite_Contact* as shown in Figure 5.7 (b).

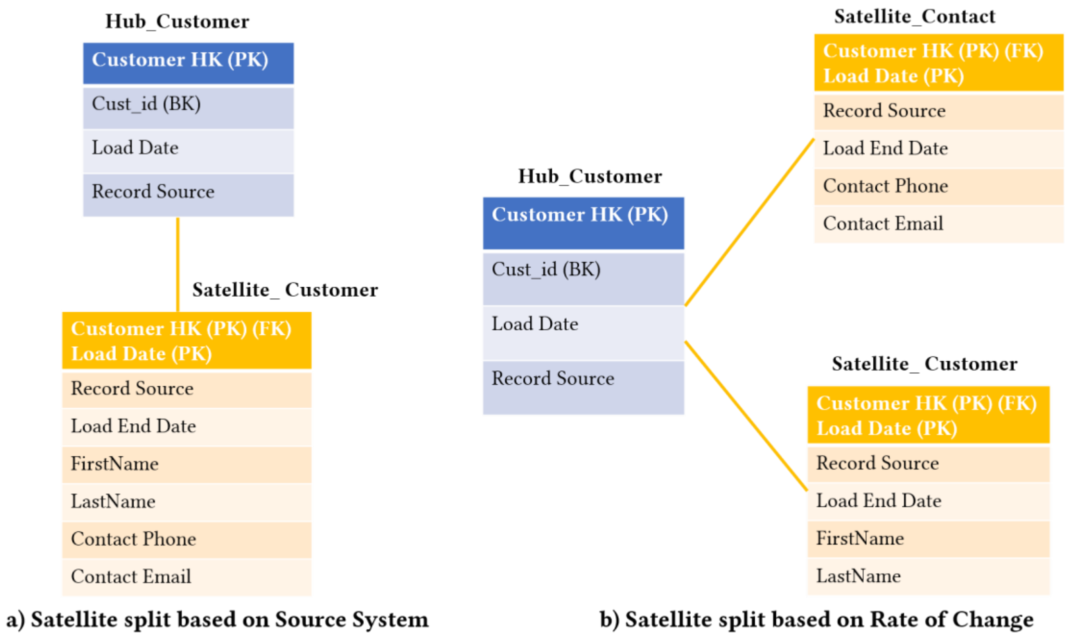


Figure 5.7: For JSON Data - Satellite Entity is Created According to TR4

JSON Arrays

JSON array[3] represents the ordered collection of values of various datatypes like the number, string, object, etc. JSON arrays are needed to group the data together when we are dealing with a large amount of data. Array values are represented between "[" and "]". If there are more values, then they are separated by ",". As shown in Figure 5.8, *Contact* array has two objects, and each of these objects have properties *Phone* and *Email*. These objects are ordered, i.e. in the *Contact* array, the first object is indexed "zero" and the second object is indexed "one". The *Contact* array values are represented between "[" and "]" and the values are separated by ",".

3 https://www.json.org/index.html

```
Customer document
{
        Cust_id: 001234
        FirstName: "Thomas",
        LastName: "George",
        Contact: [
        {
                Phone: "0567832",
                Email: "t.george@gmail.com"
        },
        {
                Phone: "0932582",
                Email: "t.george_office@abc.com"
        }
        ]
}
```

Figure 5.8: Example for JSON Array

While arrays are an essential part of the JSON structure, there are no details discussed in the paper [CJJ18] on how to handle the JSON array based on these translation rules.

5.2 Xml Schema into the Data Vault

XML makes use of own tags to describe the data. In XML documents structure, content and semantics are defined with the help of XML Schema Definition (XSD). XSD is similar to Document Type Definition (DTD), but it has more control over the XML structure. XML documents refer to DTD or XSD to define its structure.

Curtis et al. have described the process involved in the creation of an XML schema from the Data Vault model [KJ13]. Curtis et al. paper define the schema to link the Data Vault model and fully-temporalized database. Temporal databases are used to store the data, which changes over time. Valid time and the transaction time are the two different time notations used in the temporal database. Valid time denotes the time period, when the data are considered as real whereas transaction time means the time period of data are stored in the database [PG12]. For example, if *David* worked in a *company X* from *1990-01-01* to *2000-12-01*, if this information was entered in the database on *1990-05-01* and this data was deleted from the database on *2001-02-15*, then this time period is the transaction time. This can be represented in the temporal database as shown in Table 5.1. The temporal data, which are loaded into the data warehouse is addressed by the Data Vault modeling technique. To exchange the temporalized data in structured XML format with different systems, the Data Vault metamodel was generated, and it is mapped with a general XML schema.

Name	Company	Valid Time	Transaction Time
David	X	1990-01-01 to 2000-12-01	2000-12-01 to 2001-02-15
David	Y	2001-01-03 to 2013-12-01	2001-02-15 to 9999-12-30

Table 5.1: Example for Temporal Database

The components of the Data Vault metamodel needs to be identified before developing the standard XSD for the Data Vault schema. The components of the Data Vault model as used in Curtis et al. paper [KJ13] as follows:

- Data type and Time type represents data values and time values respectively.
- Hub, Link, and Satellite
- Hub_Satellite and Link_Satellite are the Satellites related to Hubs and Links respectively.
- Reference Table has a set of values, which are fixed and are used frequently. These tables are used to avoid unnecessary links and satellites. According to Linstedt [LO16], the reference table is another type of entity in the Data Vault. It is mainly used to store the data that is frequently used to give context and to define other business keys like standard codes and description. For example, the foreign key *State_ID* in a Satellite table *Sat_Supplier* references a primary key *State_ID* of the reference table *Reftable_States* as shown in Figure 5.11 (a), but they can also be used along with other Business Vault entities.
- Sometimes, more than one Hub, Link or reference table can appear in a relationship. These then have different roles, and these roles are realized as following like Hub Role, Reference Role, and Link Role. Hub Role is an entity that is used to identify, which Hub has associated through a Link with other Hubs. Link Role is an entity that is used to determine, which Links are linked to another Link. Reference Role is an entity that is used to identify, which reference table values are linked with a Link. A role is created, to lodge all these roles. To lodge all these roles, Role is created.

Hub, Link, Satellite, Hub_Satellite, Link_Satellite, and Reference table are the necessary components of the Data Vault as said in the paper [KJ13]. The relationships in the Data Vault model are expressed using cardinality.

XML schema document consists of various components like:

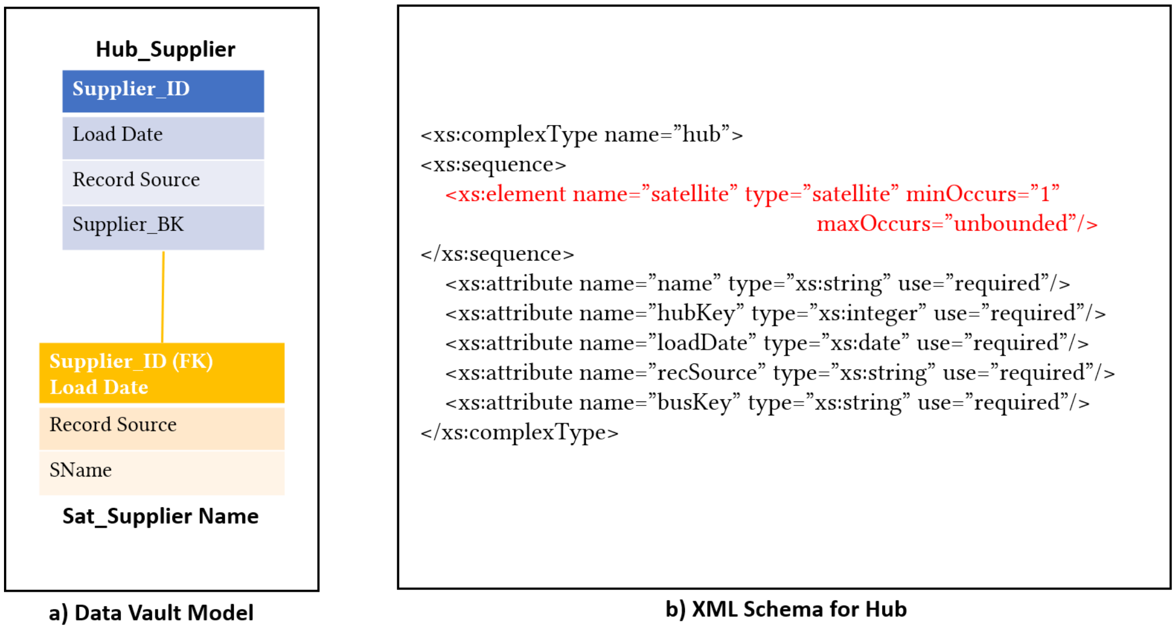
- **Tag** - It is a name which is enclosed between '<' and '>'. For example, an item is considered as a tag when it is written as *<item>*. There are two types of tags like opening tag *<item>* and the closing tag *</item>*.
- **Attributes** - It is a key-value pair represented inside the opening element. The attribute values are specified within single quotes or double quotes. Attributes are the metadata which describes an element.
- **Element** - Elements are the data which are produced by an application. It includes every content, which is represented between the opening tag and matching closing tag. Even opening and closing tag is also an element. If the element does not contain any data, it is said to be as an empty element. The elements can have text, attributes, other elements

or a combination of all. Elements have relationships, which are represented as a parent element, child element, and sibling's element. Elements will specify the type (simple or complex) and occurrences (maximum Occurrences or minimum Occurrences) when it is included in a series. The sequence states that the child elements need to appear in a sequence. The element type can be either simple or complex. If an element has only text and it does not comprise any other elements or attributes, then it represents a *simple type*. If an element contains other elements/attributes, then it is said to be as a *complex type*. By default, attributes are optional, by assigning *use= "mandatory"*, attributes can be made mandatory.

- **Namespace** - Elements or attributes in an XML document may have the same names (tag names) even though the content is different. In this case, to avoid name conflicts, XML namespace mechanism was introduced, and it is optional. The namespace is defined in the start element tag by using prefix name and *xmlns* attribute. The syntax for defining Namespace is: *<xs:schema xmlns:xs= "URI">* where xmlns denotes the XML namespace, *xs* is the namespace prefix and Uniform Resource Identifier (URI) denotes where the definition of tags are stored.

Here, we consider the essential Data Vault components and transform it into an XML schema:

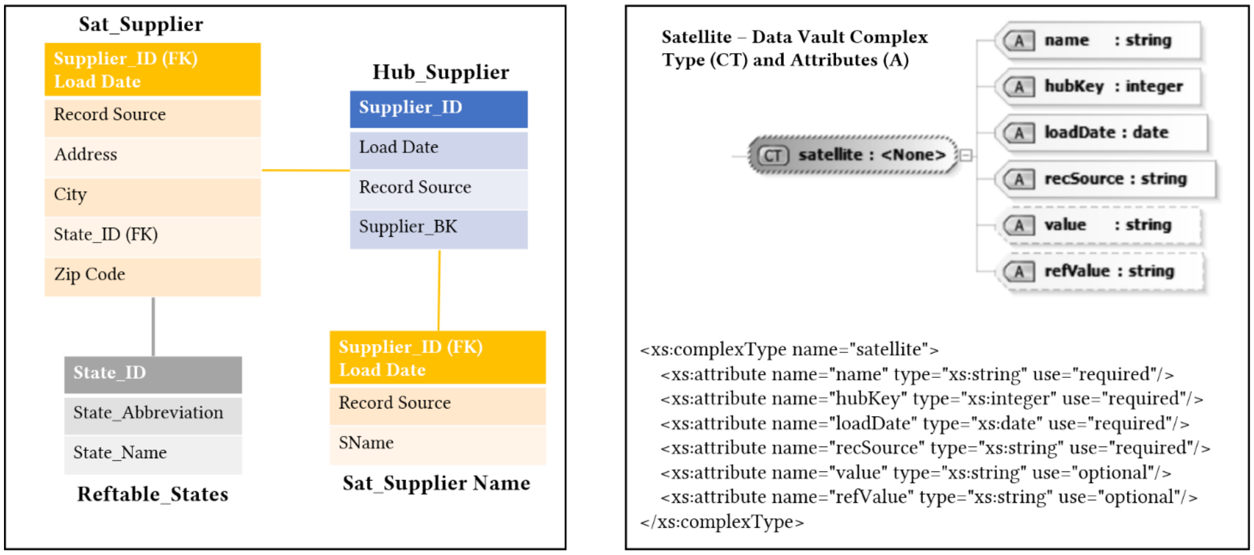
a) **Hubs** - In an XML schema, a *hub* is defined as a complex type with attributes like *name*, *hubKey*, *busKey* etc. Inside a Hub, *minOccurs="1"* and *maxOccurs="unbounded"* for a satellite *sequence* specifies that a Hub should have at least one Satellite as shown in Figure 5.9.



```
<xs:complexType name="hub">
<xs:sequence>
   <xs:element name="satellite" type="satellite" minOccurs="1"
                                                   maxOccurs="unbounded"/>
</xs:sequence>
   <xs:attribute name="name" type="xs:string" use="required"/>
   <xs:attribute name="hubKey" type="xs:integer" use="required"/>
   <xs:attribute name="loadDate" type="xs:date" use="required"/>
   <xs:attribute name="recSource" type="xs:string" use="required"/>
   <xs:attribute name="busKey" type="xs:string" use="required"/>
</xs:complexType>
```

Figure 5.9: Data Vault to XML Schema - Hub

b) **Satellites** - In an XML schema, a satellite is defined as a Complex Type (CT) with Attributes (A) like *name*, *hubkey*, *recSource*, etc. The attribute *value* denotes the string value, which is nonreferenced. i.e. *SName* in *Sat_Supplier Name*. The attribute refvalue indicates the value, which is referenced for example, *State_ID* in *Sat_Supplier*. The *value* and *refvalue* attributes are optional in Satellites, which are stated using *use="optional"* as shown in Figure 5.10.

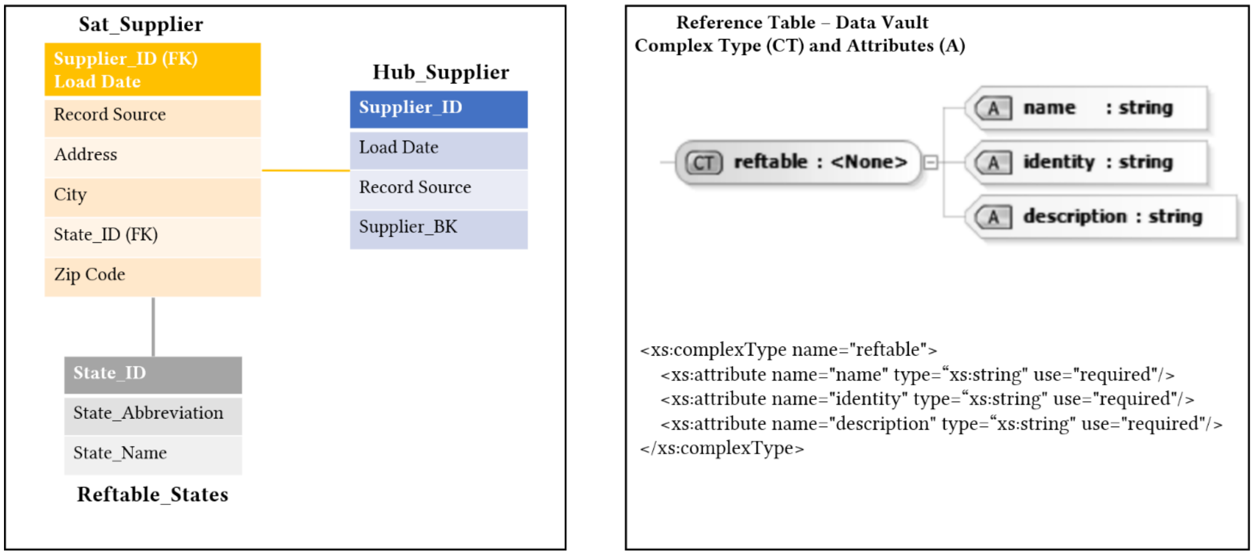


```
<xs:complexType name="satellite">
   <xs:attribute name="name" type="xs:string" use="required"/>
   <xs:attribute name="hubKey" type="xs:integer" use="required"/>
   <xs:attribute name="loadDate" type="xs:date" use="required"/>
   <xs:attribute name="recSource" type="xs:string" use="required"/>
   <xs:attribute name="value" type="xs:string" use="optional"/>
   <xs:attribute name="refValue" type="xs:string" use="optional"/>
</xs:complexType>
```

a) Data Vault Model **b) XML Schema for Satellite**

Figure 5.10: Data Vault to XML Schema – Satellite

c) **Reference Table** - In Data Vault model, a Satellite *Supplier* has a reference table *Reftable_States*, which stores the frequently accessed static data. The reference table has attributes like *State_ID*, *State_ Abbreviation* and *State_Name*. In XML schema, *reftable* is defined as a CT and their attributes (A) like *name*, *identity* etc. are defined as string type and *use="required"* specifies that the attribute is mandatory as shown in Figure 5.11.



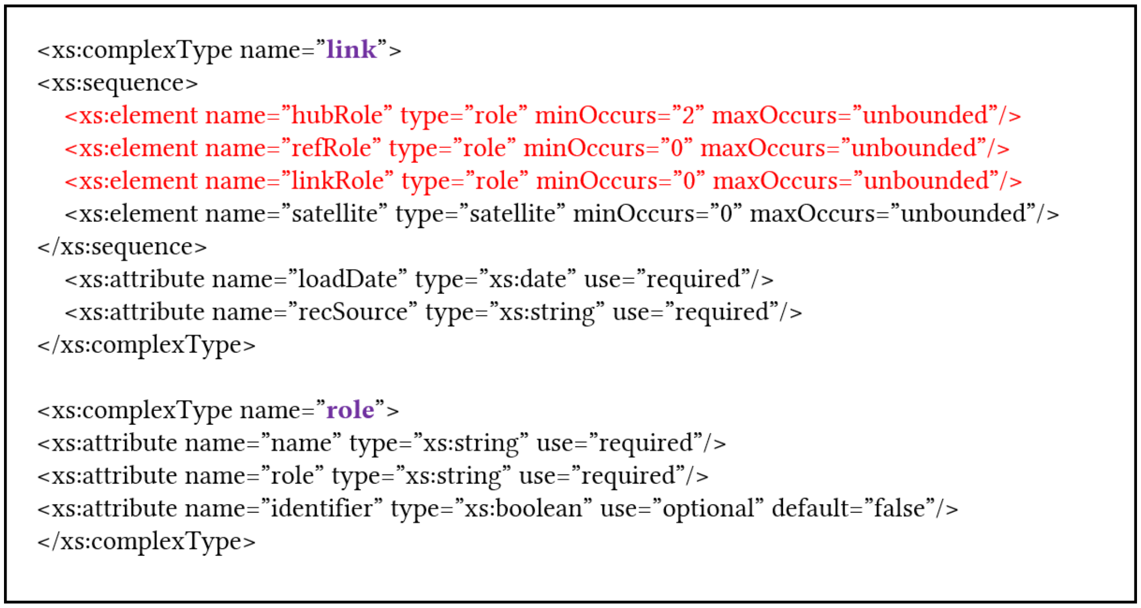
```
<xs:complexType name="reftable">
   <xs:attribute name="name" type="xs:string" use="required"/>
   <xs:attribute name="identity" type="xs:string" use="required"/>
   <xs:attribute name="description" type="xs:string" use="required"/>
</xs:complexType>
```

a) Data Vault Model **b) XML Schema for Reference Table**

Figure 5.11: Data Vault to XML Schema - Reference Table

d) **Links** - In XML schema a *link* has *hubRole*, *refRole* and *linkRole* as shown in Figure 5.12. Inside the *hubRole* sequence *minOccurs="2"* and *maxOccurs="unbounded"* specifies that a Link should have at least two Hub elements whereas *refRole/linkRole* elements can have zero or any number in a Link. Along with these elements, Links have attributes like *loadDate* and *recSource*. To fit Hub Roles, Link Roles and Reference Roles,

   Role was defined as complex type. *Role* has attributes like *name*, *role* and *identifier*. The *name* denotes the component name, which is modeled; *role* specifies the relationship of the component with other components in a Link. The *identifier* is an optional attribute with type as *boolean*, indicating either true or false based on the fact if the object is a part of the unique identifier or not in a link.



```
<xs:complexType name="link">
<xs:sequence>
   <xs:element name="hubRole" type="role" minOccurs="2" maxOccurs="unbounded"/>
   <xs:element name="refRole" type="role" minOccurs="0" maxOccurs="unbounded"/>
   <xs:element name="linkRole" type="role" minOccurs="0" maxOccurs="unbounded"/>
   <xs:element name="satellite" type="satellite" minOccurs="0" maxOccurs="unbounded"/>
</xs:sequence>
   <xs:attribute name="loadDate" type="xs:date" use="required"/>
   <xs:attribute name="recSource" type="xs:string" use="required"/>
</xs:complexType>

<xs:complexType name="role">
<xs:attribute name="name" type="xs:string" use="required"/>
<xs:attribute name="role" type="xs:string" use="required"/>
<xs:attribute name="identifier" type="xs:boolean" use="optional" default="false"/>
</xs:complexType>
```

Figure 5.12: XML Schema for Link and Role

This paper [KJ13] gives an idea of how an XML schema is formed from the Data Vault model. Next step is to think of how to transform the XML data into the Data Vault model.

6 Proposed Ideas to Integrate Semi-Structured and Unstructured Data into the Data Vault Model

The previous chapter provides the insight on the possibilities to integrate semi-structure data into the Data Vault model, but it does not cover everything on semi-structured data. Therefore, this chapter presents the newly developed approaches to integrate semi-structured data and unstructured data into the Data Vault model.

6.1 Approaches to Integrate Semi-Structured Data into the Data Vault Model

In this Subsection, we will discuss how to integrate JSON array data into the Data Vault model and how to integrate XML data into the Data Vault model.

6.1.1 JSON Array data in the Data Vault

Cernjeka et al. have explained the translation rules (TR0 to TR4) to integrate JSON document to the Data Vault model, but there are no rules to describe how to manage the array structure in JSON document [CJJ18]. As an extension of that work, we propose Translation Rule (TR5) to handle the JSON array data.

Translation Rule (TR5): JSON array structure is translated to the Data Vault model based on Multi-Active Satellite. In this case, TR1 to TR4 are applied accordingly.

JSON array data can be handled with one of the Satellite applications called "*Multi-Active Satellites*" [LO16] [L18]. Multi-Active Satellites are used to store multiple child values, and the child values do not have the business key on its own. Some examples are customer having multiple addresses, multiple phone numbers, which are active at the same time and these records are descriptive attributes. In this case, add a *sequence identifier/number* as an ordering attribute, which acts as an index to identify the multiple values in a Satellite, and it is made as a part of a primary key. As shown in Figure 6.3 and Table 6.1, *Sat_Customer Contact* explains the use of the *Sequence Number* in Multi-Active Satellites. For example, the JSON document has an id field *Customer_id* and non-id field *Customer Name*. In addition to that, it has a *Contact* JSON array data embedded to a parent document. Based on this information:

Step 1 is to apply **TR1** and create a Data Vault Hub with the business key *Customer_id* and a hash key *Customer HK,* which is calculated from the business key as shown in Figure 6.1.

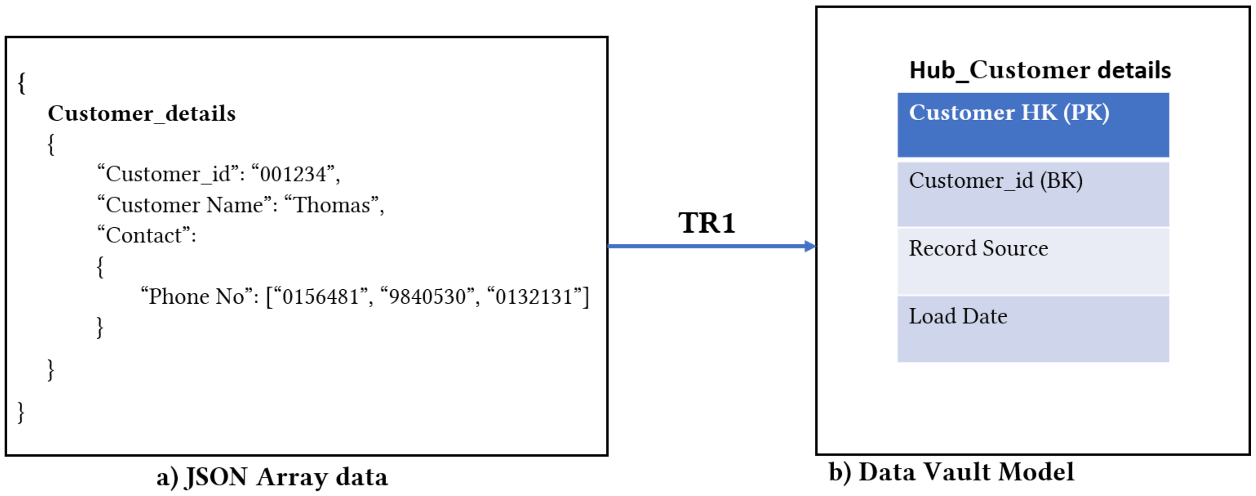


Figure 6.1: For JSON Array Data - Hub Entity is Created According to TR1

Step 2 is to apply **TR2** and create a Data Vault Satellite with a descriptive attribute *Customer Name* as shown in Figure 6.2.

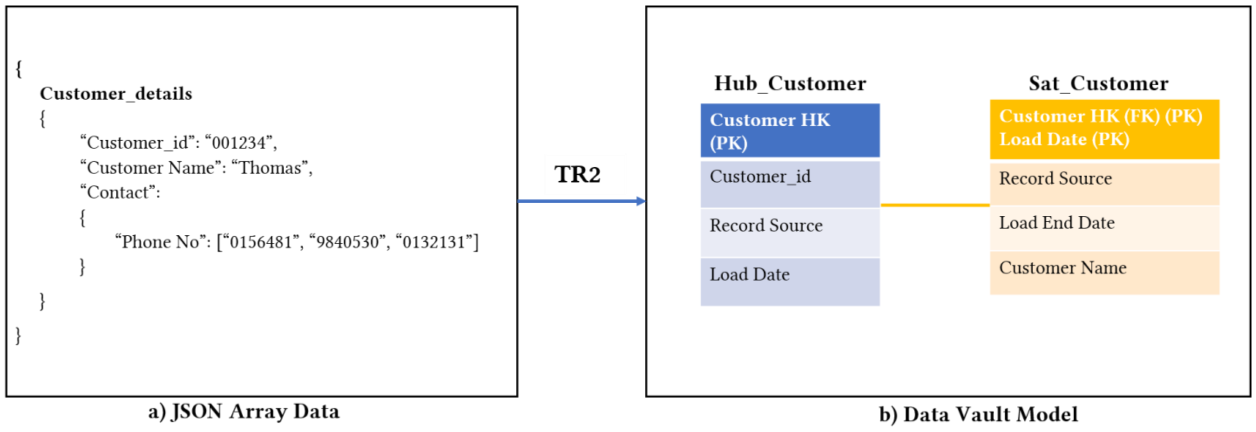


Figure 6.2: For JSON Array Data - Satellite Entity is Created According to TR2

Step 3 is to handle the embedded document and according to **TR4**, the embedded document can be translated into a Data Vault Satellite. However, in this case, the embedded document has array data *Phone No*. Therefore, we apply **TR5**, and those entries that are in the array get their Satellite, as a Multi-Active Satellite as shown in Figure 6.3. The *Sequence Number* along with other primary keys *Customer HK* and *Load Date* act as an index in a Satellite *Sat_Customer Contact* to identify multiple phone numbers as shown in Table 6.1.

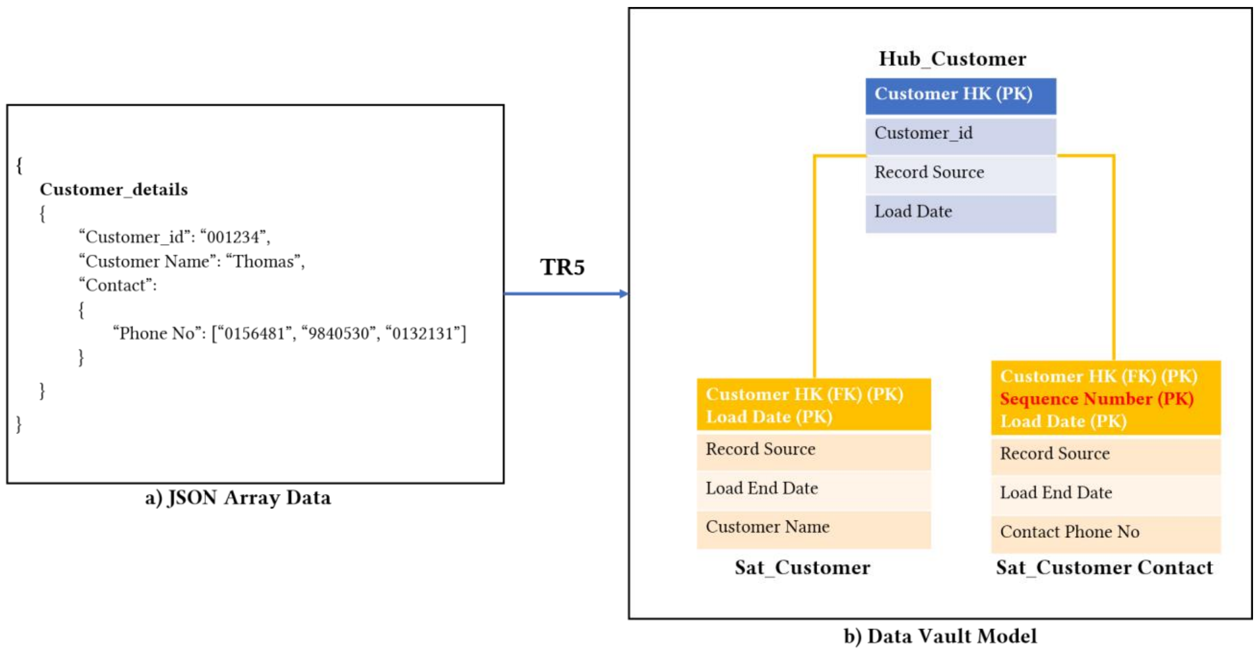


Figure 6.3: For JSON Array Data - Multi-Active Satellite Entity is Created According to TR5

a)

Satellite_Customer

Customer HK (PK)	Load Date (PK)	Record Source	Load End Date	Customer Name
56r111a...	2017-01-21 08:10:15	Customer.json	2018-01-21 09:10:15	Thomas

b)

Satellite_Customer Contact

Customer HK (PK)	Sequence Number (PK)	Load Date (PK)	Record Source	Load End Date	Contact Phone No
56r111a...	1	2017-01-21 08:10:15	Customer.json	9999-12-30 24:00:00	0156481
56r111a...	2	2017-01-21 08:10:15	Customer.json	9999-12-30 24:00:00	9840530
56r111a...	3	2017-01-21 08:10:15	Customer.json	9999-12-30 24:00:00	0132131

Table 6.1: Satellite Data and Multi-Active Satellite Data

6.1.2 XML Data Modeled into the Data Vault

Knowles et al. have explained how to map the Data Vault to an XML schema [KJ13]. As an extension of this work, we propose the idea to integrate an XML data into the Data Vault model.

XML data are stored as a text entity or an attribute of an XML node, whereas JSON data are stored as key-value pairs [N13]. XML documents are said to be well-formed when it has a single root element and when it is properly nested (every opening tag has a matching closing tag).

The proposed idea is to integrate XML data into the Data Vault model using these two steps:

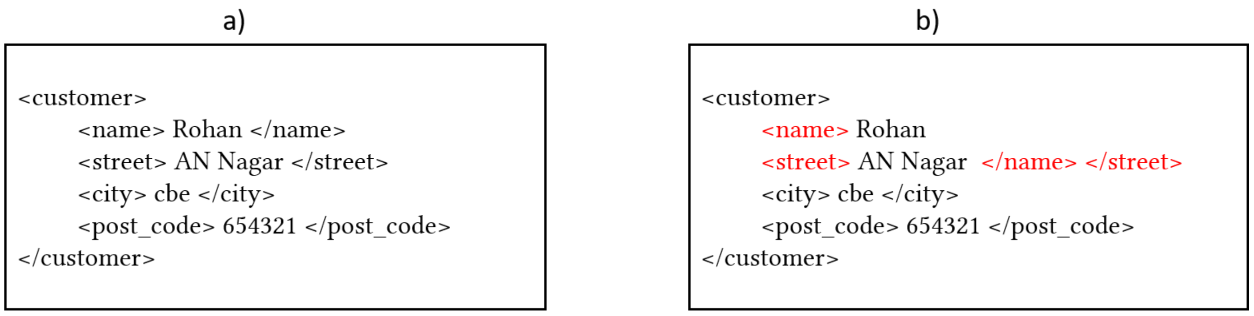
Step 1: Apply the transformation rules to transform an XML document to JSON data.
Step 2: Apply the translation rules to transform JSON document into the Data Vault model.

Step 1: Transform the XML data to JSON data based on the transformation rules

A set of rules has to be followed to transform XML data into JSON data based on various references [KJ13] [N13] [BGM+11].

Rule 1: XML structure needs to be preserved when transforming the data into JSON.

Figure 6.4 (a) is a well-formed XML document because it has a single root element *customer* and all the opening and closing tags are properly nested. Figure 6.4 (b) is not a well-formed XML document because the closing tag of *name* is not properly nested. So according to Rule 1, Figure 6.4 (a) can be transformed into JSON data.


a)

```
<customer>
        <name> Rohan </name>
        <street> AN Nagar </street>
        <city> cbe </city>
        <post_code> 654321 </post_code>
</customer>
```

b)

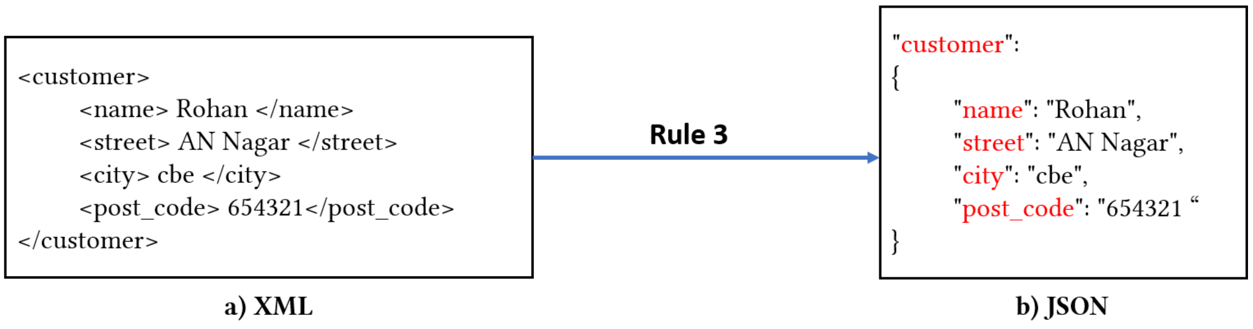
```
<customer>
        <name> Rohan
        <street> AN Nagar  </name> </street>
        <city> cbe </city>
        <post_code> 654321 </post_code>
</customer>
```

Figure 6.4: XML Document a) Well-formed b) Not Well-formed

Rule 2: XML namespace is not supported in JSON.

Rule 3: Element or attribute names in an XML are converted to JSON object names.

According to Rule 3, XML elements like *customer*, *name*, *street* are represented as JSON object names within double quotes as shown in red in Figure 6.5. Similarly, XML attribute names like *numbe*r and *date* are converted to JSON object names are shown in red color in below Figure 6.6. The symbol '*@*' is used to differentiate between the attribute name and element name in JSON.

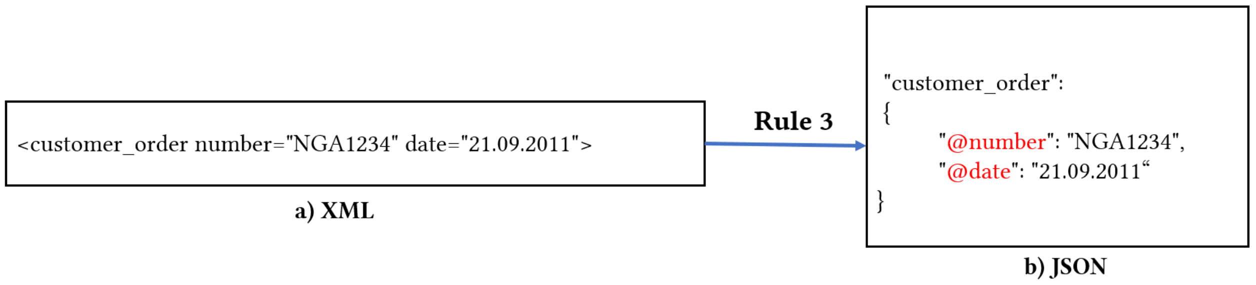


```
<customer>
        <name> Rohan </name>
        <street> AN Nagar </street>
        <city> cbe </city>
        <post_code> 654321</post_code>
</customer>
```

```
"customer":
{
        "name": "Rohan",
        "street": "AN Nagar",
        "city": "cbe",
        "post_code": "654321 "
}
```

a) XML **b) JSON**

Figure 6.5: Rule 3 to Transform XML Element Names to JSON Object Names



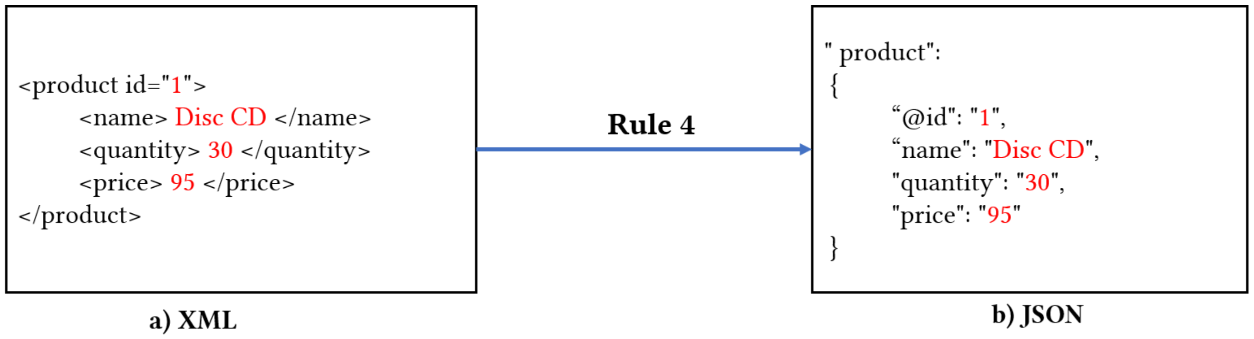
```
<customer_order number="NGA1234" date="21.09.2011">
```

```
"customer_order":
{
        "@number": "NGA1234",
        "@date": "21.09.2011“
}
```

Figure 6.6: Rule 3 to Transform XML Attribute Names to JSON Object Names

Rule 4: Text nodes in an XML are changed into JSON simple values.

According to Rule 4, text nodes of elements like *Disc CD*,*30* and attributes like *1* in an XML, are represented as JSON values within double quotes as shown in red in Figure 6.7.



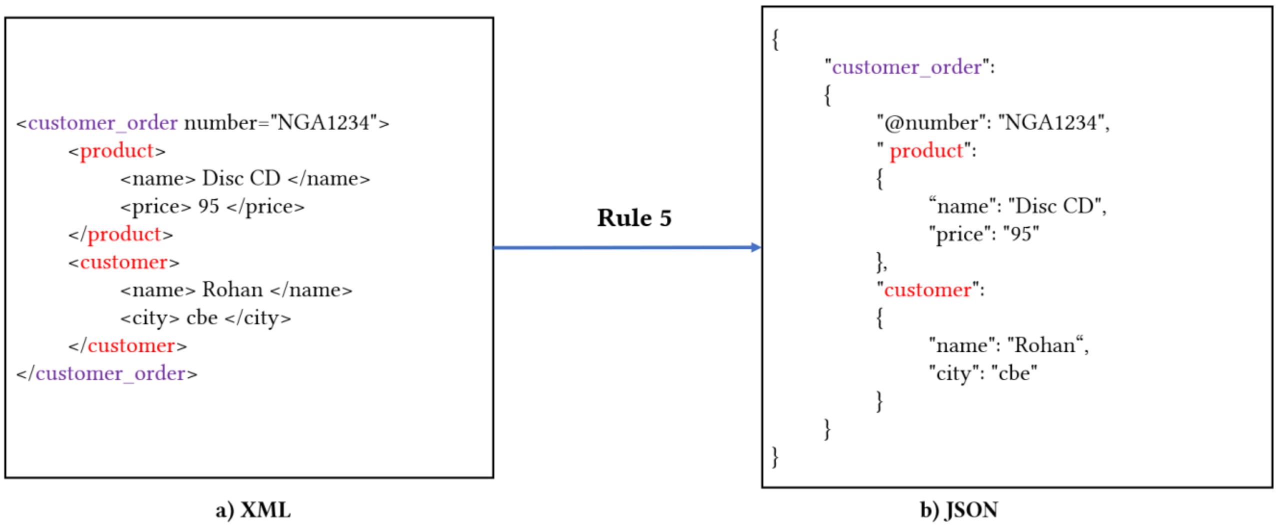
```
<product id="1">
        <name> Disc CD </name>
        <quantity> 30 </quantity>
        <price> 95 </price>
</product>
```

```
" product":
{
        “@id": "1",
        “name": "Disc CD",
        "quantity": "30",
        "price": "95"
}
```

Figure 6.7: Rule 4 to Transform Text Nodes to JSON Values

Rule 5: The child elements in an XML with the same parent will become JSON object fields or names.

As shown in Figure 6.8, *customer_order* is the parent or root element for the child elements *product* and *customer*. In other words, elements *product* and *customer* have the same parent element *customer_order*. Likewise, the parent element *product* has child elements like *name* and *price*. According to Rule 5, the child elements with the same parent are transformed to JSON object names as shown in red in Figure 6.8.



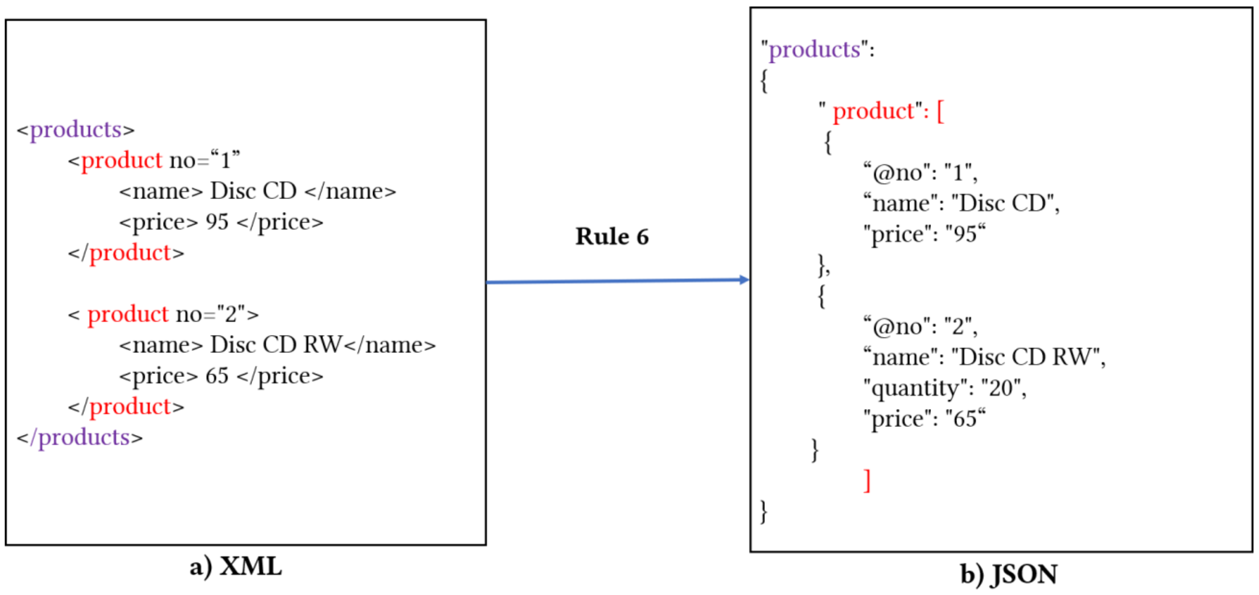
```
<customer_order number="NGA1234">
        <product>
                <name> Disc CD </name>
                <price> 95 </price>
        </product>
        <customer>
                <name> Rohan </name>
                <city> cbe </city>
        </customer>
</customer_order>
```

```
{
        "customer_order":
        {
                "@number": "NGA1234",
                " product":
                {
                        "name": "Disc CD",
                        "price": "95"
                },
                "customer":
                {
                        "name": "Rohan",
                        "city": "cbe"
                }
        }
}
```

Figure 6.8: Rule 5 to Transform the Child Elements to JSON Object Names

Rule 6: XML data with multiple child elements with the same name are transformed to JSON array elements.

As shown in Figure 6.9, an XML parent element *products* has multiple child elements with same name *product*. These child elements are represented as JSON array element *product* between opening '['and closing ']'.



```
<products>
        <product no="1"
                <name> Disc CD </name>
                <price> 95 </price>
        </product>

        < product no="2">
                <name> Disc CD RW</name>
                <price> 65 </price>
        </product>
</products>
```

```
"products":
{
        " product": [
         {
                "@no": "1",
                "name": "Disc CD",
                "price": "95"
         },
         {
                "@no": "2",
                "name": "Disc CD RW",
                "quantity": "20",
                "price": "65"
         }
                ]
}
```

Figure 6.9: Rule 6 to Transform Multiple Child Elements with the Same Name to JSON Array Elements

Step 2: Apply the Translation Rules to transform the JSON document to the Data Vault model

The next step is to apply the translation rules (TR0 to TR5) to transform JSON document to the Data Vault model as discussed earlier. Here, we will discuss how to use the translation rules accordingly to transform some XML documents into the Data Vault model.

Example 1

For instance, an XML document, which has an id field as the *number* and has child elements like *product* and *customer* data, which are embedded within the *customer_order* document. Based on this information:

Step 1 is to transform an XML element like *customer_order* and attribute like *number* to JSON object names and then apply **TR1** to create a Data Vault Hub with the business key *Customer number* and a hash key *Customer HK* is calculated from the business key as shown in Figure 6.10.

From this, we can deduce that an XML element *customer_order*, which has an id attribute *number* can be transformed to a Data Vault Hub *Hub_customer_order*. The attribute which is unique forms the business key *Customer number*.

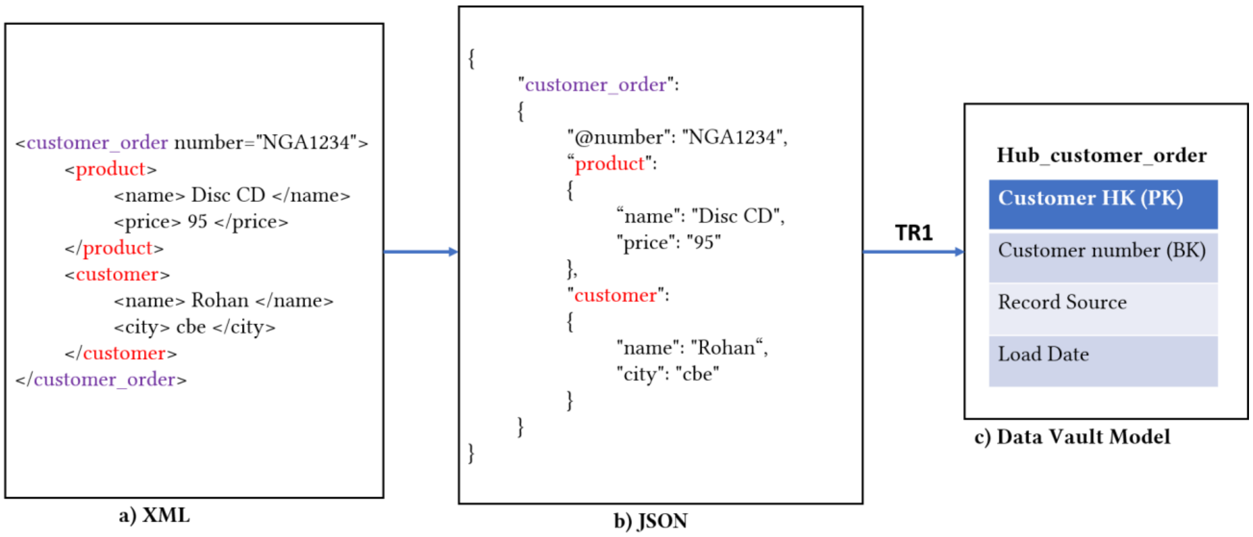

```
<customer_order number="NGA1234">
        <product>
                <name> Disc CD </name>
                <price> 95 </price>
        </product>
        <customer>
                <name> Rohan </name>
                <city> cbe </city>
        </customer>
</customer_order>
```
```
{
        "customer_order":
        {
                "@number": "NGA1234",
                “product":
                {
                        “name": "Disc CD",
                        "price": "95"
                },
                "customer":
                {
                        "name": "Rohan“,
                        "city": "cbe"
                }
        }
}
```

Figure 6.10: XML to JSON to Data Vault- Hub Entity is Created According to TR1

Step 2 is to transform XML child elements of *customer_order* like the *product*, *customer* and grandchild elements of *customer_order* like *price*, *city*, etc. as JSON object names. Then apply **TR4** and translate the embedded sub-documents like the *product*, *customer* to a Data Vault Satellite *Sat_customer_order*, which are connected to the parent Hub *Hub_customer_order* as shown in Figure 6.11.

From this, we can deduce that XML child elements of *customer_order* like *product* and *customer* are transformed to a Data Vault Satellite *Sat_customer_order* if they do not have an id field. Then, XML child elements with text node like *name*, *price* form the descriptive attributes of a Satellite.

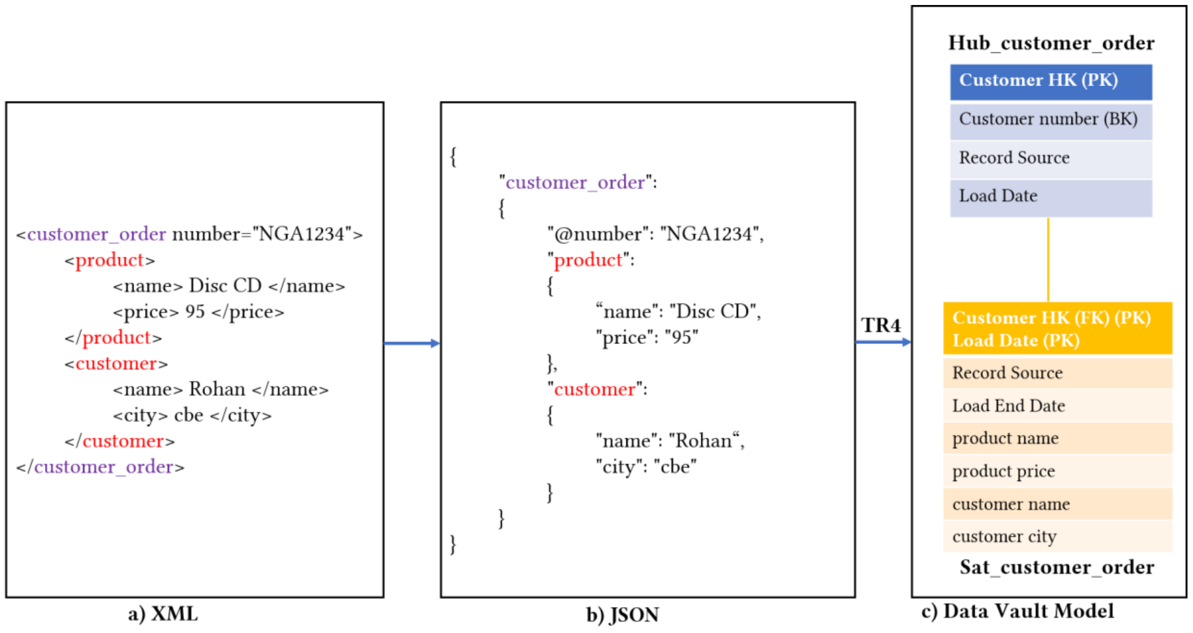


Figure 6.11: XML to JSON to Data Vault- Satellite Entity is Created According to TR4

Example 2

For instance, an XML document, which has an id field as *number* and XML child elements with same name *product*. Based on this information:

Step 1 is to transform XML elements like *customer_order*, *product*, *name*, etc. and attribute of *customer_order* like *number* to JSON object names. Later apply **TR1** and create a Data Vault Hub with the business key *Customer number* and a hash key is calculated from the business key as shown in Figure 6.12.

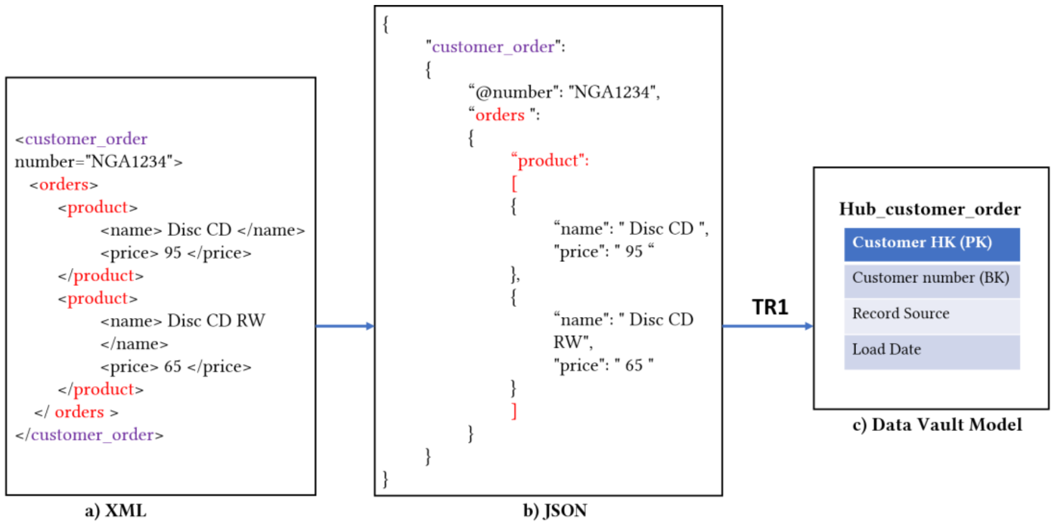
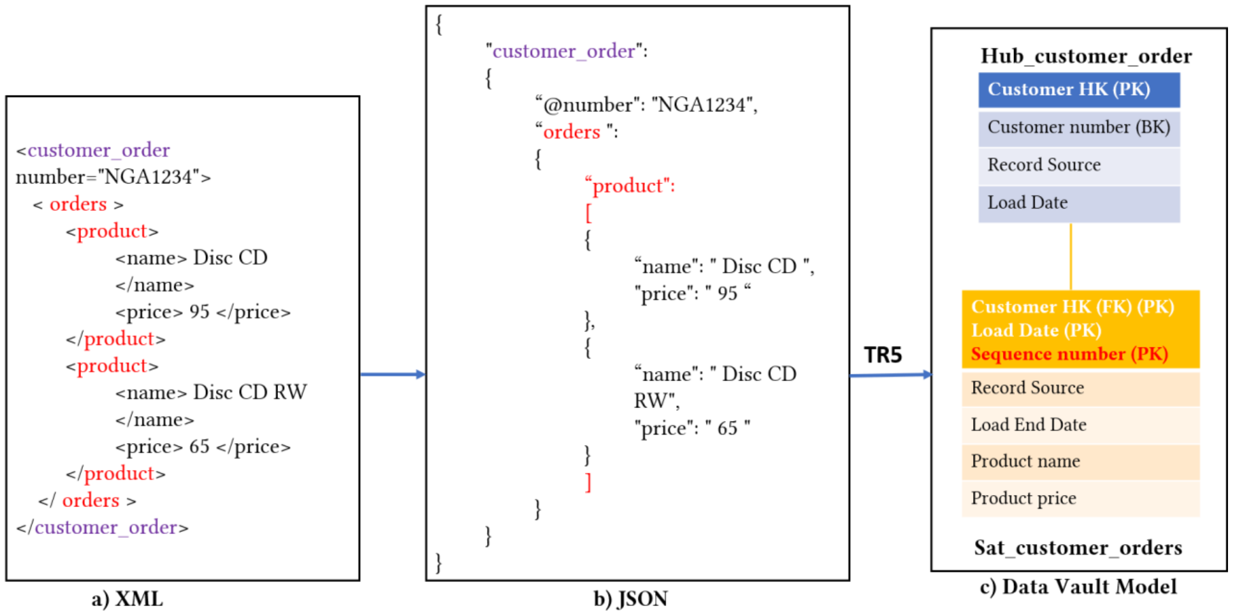


Figure 6.12: XML to JSON Array Data to Data Vault- Hub Entity is Created According to TR1

Step 2 is to transform multiple child elements with the same name into JSON array elements. For example, *orders* element has various child elements with same name *product* are changed to JSON array element. Later apply **TR5** and translate the embedded array data *product* into a Data Vault Satellite based on Multi-Active satellites as shown in Figure 6.13.

From this, we can deduce that XML child elements that do not have an id field and have the same element name like product can be transformed to a Data Vault Multi-Active Satellite *Sat_customer_orders.*



```
<customer_order
number="NGA1234">
  < orders >
      <product>
            <name> Disc CD
            </name>
            <price> 95 </price>
      </product>
      <product>
            <name> Disc CD RW
            </name>
            <price> 65 </price>
      </product>
   </ orders >
</customer_order>
```

```
{
      "customer_order":
      {
            "@number": "NGA1234",
            "orders ":
            {
                  "product":
                  [
                  {
                        "name": " Disc CD ",
                        "price": " 95 "
                  },
                  {
                        "name": " Disc CD
                        RW",
                        "price": " 65 "
                  }
                  ]
            }
      }
}
```

Figure 6.13: XML to JSON Array Data to Data Vault- Hub Entity is Created According to TR5

Till now we have seen, how an XML document can be transformed into the Data Vault model with the help of intermediate steps i.e., XML to JSON and JSON to the Data Vault model. Now we will define the rules that allow us to directly transform an XML document to the Data Vault model.

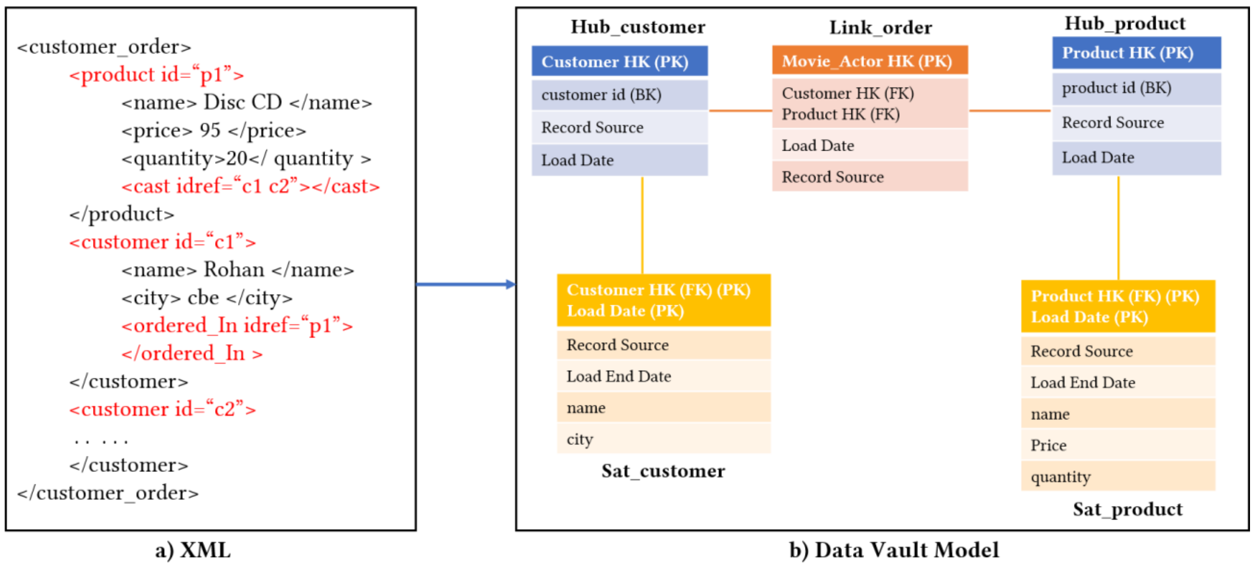
Example 3

For instance, an XML document, which has attributes like ID and IDREF (ID Reference). The ID is used to identify elements, and IDREF is used to refer to other elements. As shown in Figure 6.14:

Step 1: To transform XML child elements *product*, *customer* with attribute *id* to Hubs namely *Hub_product* and *Hub_customer*.

Step 2: XML elements having attribute types like ID and IDREF are transformed into a Data Vault Link. XML child elements having attribute name *idref* with values *c1* and *c2* that refers to ID attribute name and value *customer id c1* and *customer id c2* are transformed to a Data vault Link *Link_order*.

Step 3: XML child elements of customer_order like the *product*, *customer* are transformed into a Data Vault Satellites *Sat_customer* and *Sat_product*. Then, XML child elements with text node like *name*, *price*, *city* form the descriptive attributes of a Satellite.



```
<customer_order>
       <product id="p1">
              <name> Disc CD </name>
              <price> 95 </price>
              <quantity>20</ quantity >
              <cast idref="c1 c2"></cast>
       </product>
       <customer id="c1">
              <name> Rohan </name>
              <city> cbe </city>
              <ordered_In idref="p1">
              </ordered_In >
       </customer>
       <customer id="c2">
       . .  . . .
       </customer>
</customer_order>
```

Figure 6.14: XML to Data Vault Link Entity According to Rule 4

From the above examples, we can infer the following rules to transform XML data into the Data Vault Model:

Rule 1: An XML first element that is highest in the hierarchy and has an id attribute is transformed into a Data Vault Hub. The id attribute forms the business key.

Rule 2: XML child elements without an id attribute can be transformed into a Data Vault Satellite, and the text nodes (for example, in Figure 6.14 like *Disc CD*, *95*) form the descriptive attributes of a Satellite.

Rule 3: If XML child elements are embedded and if they have a unique id field, then they are transformed into Hubs, and a Data Vault Link is added between a Hub of the parent XML element and a Hub of the embedded element.

Rule 4: XML multiple child elements with the same name can be transformed into a Data Vault Multi-Active Satellite.

Rule 5: XML document elements using attribute types of ID and IDREF are transformed to a Data Vault Link. References between XML elements (indicated by the attribute types ID or IDREF) are transformed into a Data Vault Link between the respective Hubs.

6.2 Approaches to Integrate Unstructured Data into the Data Vault Model

Before we discuss the various approaches to integrate unstructured data into the Data Vault model that are developed in this thesis, it is necessary to understand the problems in unstructured data and how they can be solved.

Problem: For unstructured data, the filename is the only possibility to have as the business key, as it helps to identify the file uniquely. However, it cannot be considered as the business key because the name of the file is dynamic. Therefore, unstructured data like images, videos and audios do not have a proper business key [Lin17].

Solution: As discussed earlier in the Subsection 4.1.1, in some cases these composite/smart key provides a solution to this problem. The composite business key is a combination of more than one business key. For example, if we want to store the image of the product, the composite business key can be formed by combining the business key of a product, i.e. *Product ID (PID)* with the business key of the image, i.e*. Image ID (ImID)*. Thus, more than one fields are combined to form the composite business key i.e. *PID_ImID* for the product image as shown in Table 6.2.

Product ID (PID)	Image ID (ImID)	PID_ImID
P001	IMG01	P001_IMG01
P002	IMG02	P002_IMG02

Table 6.2: Composite Business Key for Image

However, in some cases, the image might not have the business key on its own. This might happen when the image is sent as an attachment in an email. In this case, the email has a unique business key, i.e. *Email id (EmID)* but the image does not have the business key on its own. In this case, a composite business key cannot be formed as image do not have the business key on its own i.e., Image exists only in the context of an email.

To discuss the various approaches, here we consider unstructured data like images/videos/ audio (in Subsection 6.2.1) and IoT data (in Subsection 6.2.2), which are produced through various use cases. The overview of the several approaches to integrate unstructured data into the Data Vault model is shown in Figure 6.15.

6.2.1 Image/ Video/ Audio Data

For simplicity, we will consider the image data in the following discussion. However, the approaches provided below are also suitable for audio and video data.

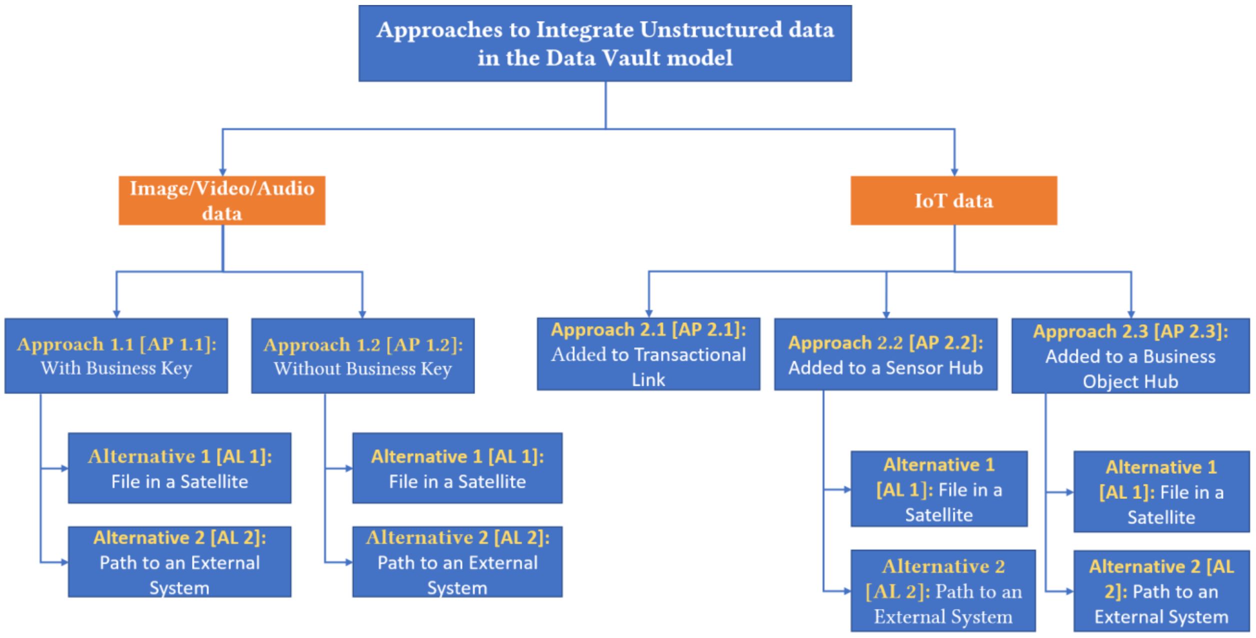
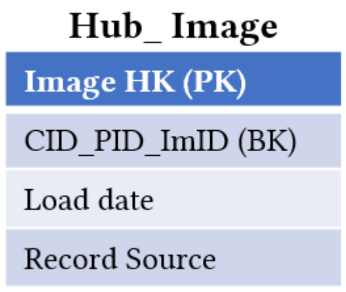

Approaches to Integrate Unstructured data in the Data Vault model
Image/Video/Audio data
IoT data
Approach 1.1 [AP 1.1]: With Business Key
Approach 1.2 [AP 1.2]: Without Business Key
Approach 2.1 [AP 2.1]: Added to Transactional Link
Approach 2.2 [AP 2.2]: Added to a Sensor Hub
Approach 2.3 [AP 2.3]: Added to a Business Object Hub
Alternative 1 [AL 1]: File in a Satellite
Alternative 2 [AL 2]: Path to an External System
Alternative 1 [AL 1]: File in a Satellite
Alternative 2 [AL 2]: Path to an External System
Alternative 1 [AL 1]: File in a Satellite
Alternative 2 [AL 2]: Path to an External System
Alternative 1 [AL 1]: File in a Satellite
Alternative 2 [AL 2]: Path to an External System

Figure 6.15: Various Approaches to Integrate Unstructured Data into the Data Vault Model

Approach 1.1 [AP 1.1]: Unstructured data with the business key

In some cases, unstructured data like image, audio or video has the composite business key. Therefore, it can be modeled as a Hub with a Satellite. As shown in Figure 6.16, *CID_PID_ImID* represent the business key for a *Hub_Image*.



Hub_ Image
Image HK (PK)
CID_PID_ImID (BK)
Load date
Record Source

Figure 6.16: Image Modeled as a Hub

Alternative 1 [AL 1]: File in a Satellite

These image files are stored directly in a Satellite table. As shown in Figure 6.17, the image attributes like *Image Name*, *Timestamp* and *Image File* are stored directly in a Satellite *Sat_Image*.

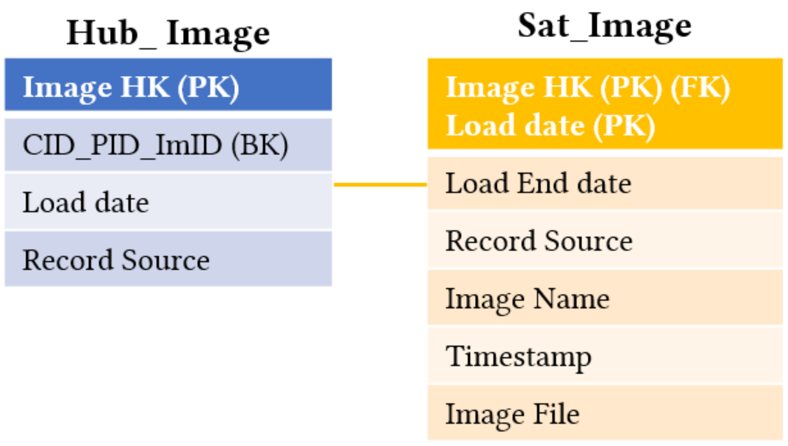


Figure 6.17: Image Files are Stored Directly in a Satellite

Alternative 2 [AL 2]: Path to an External System

These image files are not stored directly in a Satellite table, instead it is stored in an external system suited for the management of such data (e.g., Hadoop Distributes File System (HDFS)) and a Satellite contains the path to the actual file i.e., the location of the file where it is stored in an external system. As shown in Figure 6.18, the image attributes like *Image Name*, *Timestamp* are stored directly in a Satellite, and the image file is stored in the external system, and the path of that file is stored in a *Sat_Image*.

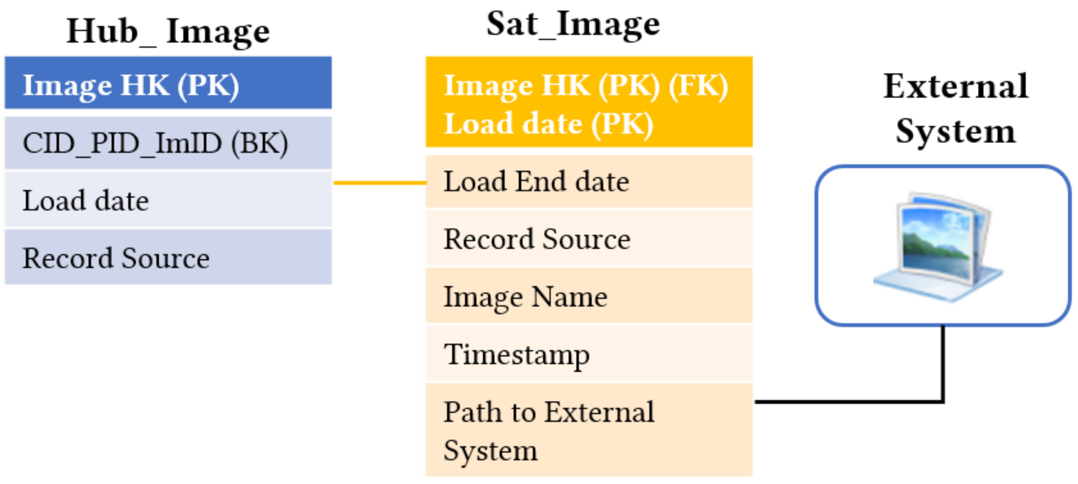


Figure 6.18: Satellite Attributes Stored Using an External Link

If we model an image as a Hub, we have the advantage of adding a Satellite to a Hub as shown in above Figure 6.18. In addition to that, we can also add new Hubs to the existing model with the help of a Link, which shows that the Data Vault model provides flexibility. As shown in Figure 6.19, *Hub_Image* can be extended to store the product image with *Hub_Product*, which is linked to existing *Hub_image* with the help of *Link_Product Image*.

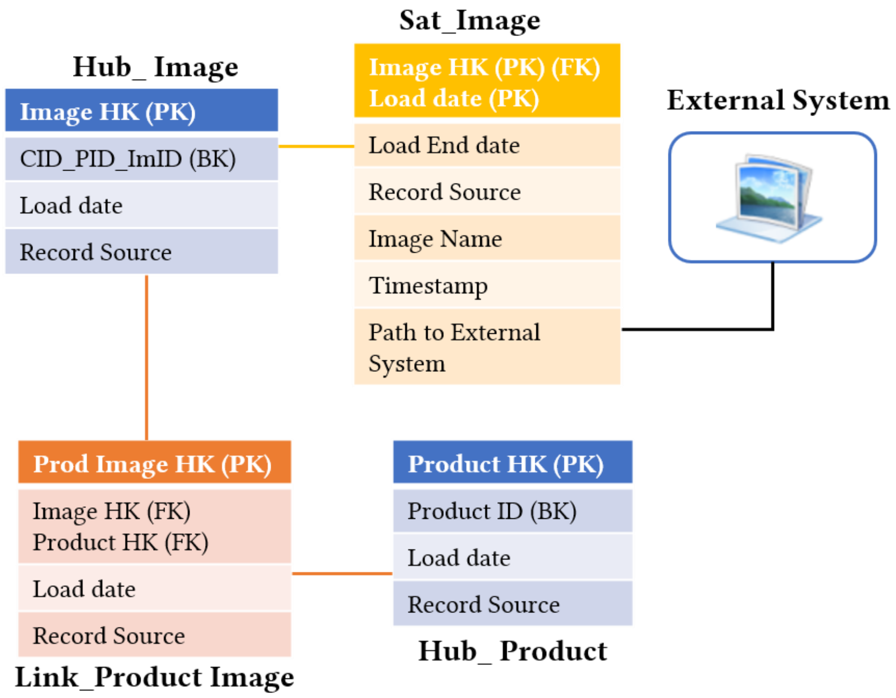


Figure 6.19: Image Hub Extended with the Help of a Link

Therefore, AP 1.1 has the great benefit that a Hub can be linked to other Hubs and also has the advantage of adding more Satellites to a Hub.

Approach 1.2 [AP 1.2]: Unstructured data without the business key

In some cases, unstructured data do not have the business key on its own. So, it can be modeled as a Satellite, which is linked to a Hub that it belongs to. For example, images are added as an attachment to an email. Here, an email is modeled as a Hub with *Email ID* as the business key, and the image, which does not have the business key is modeled as a Satellite *Sat_Image*, and it is attached to *Hub_Email* as shown in Figure 6.20.

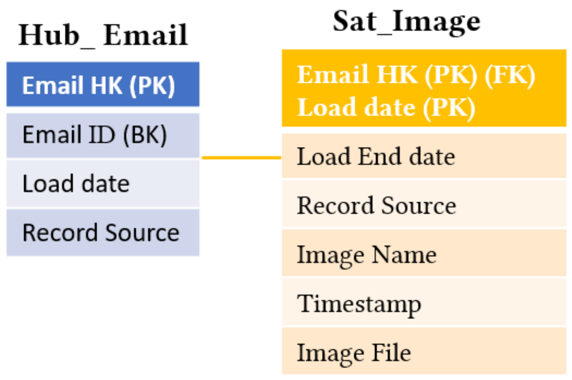


Figure 6.20: Image Modeled as a Satellite

Alternative 1 [AL 1]: File in a Satellite

These image files are stored directly in a Satellite table. As shown in Figure 6.21, the image attributes like *Image Name*, *Timestamp* and *Image File* are stored directly in a Satellite *Sat_Image*.

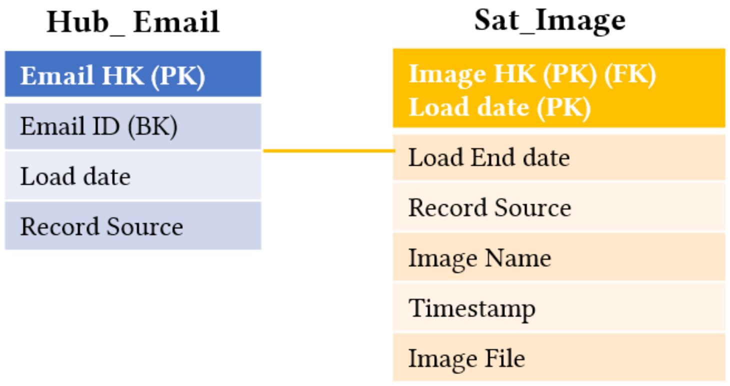


Figure 6.21: Image Files are Stored Directly in a Satellite

Alternative 2 [AL 2]: Path to an External System

These image files are not stored directly in a Satellite table. Instead, it is stored in an external system, and a Satellite contains the path to the actual file i.e., the location of the file where it is stored in an external system. As shown in Figure 6.22, email details like *From*, *To*, *Timestamp*, etc. are modeled as a Satellite namely *Sat_Email* and Image attributes *Image Name*, *Timestamp*, are modeled as a Satellite namely *Sat_Image*. The image file in the *Sat_Image* is stored in an external system, and the path of that file is stored in a *Sat_Image*.

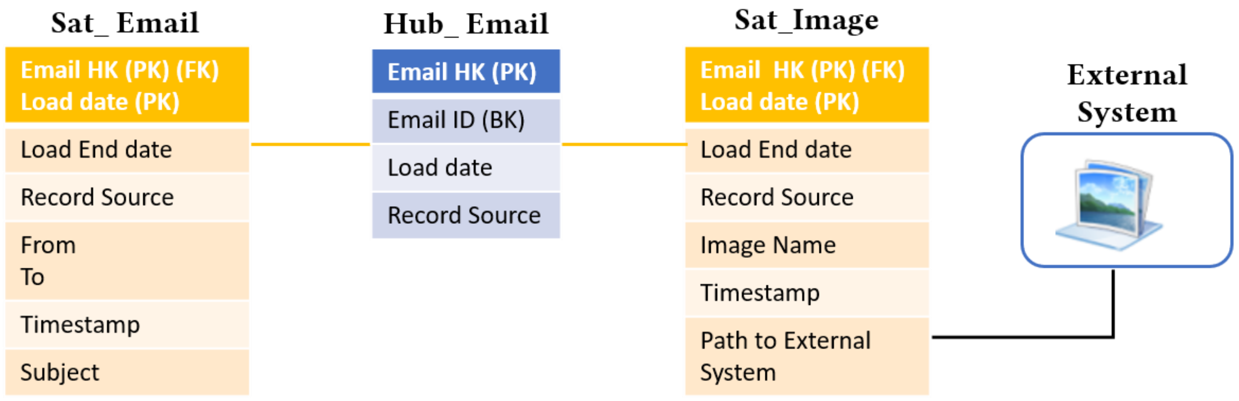


Figure 6.22: Image Files are Stored Using an External Link

In this case, if we model the image as a Satellite then we cannot extend the model i.e., we cannot link it to other Hubs. So, it is beneficial to model the image as a Hub as said in AP 1.1 to support the flexibility in the Data Vault model.

6.2.2 IoT data

IoT data enables supervising the production process and plays a significant role in the Autonomous Car Testing use case.

There are various ways how these unstructured data (IoT) data can be stored in the Data Vault model and these approaches are explained below.

Approach 2.1 [AP 2.1]: IoT data added to Transactional Link

Transactional Link is one of the applications of a Link. Transactional link also known as NonHistorized link, states there are two ways to model transactional links in the Data Vault model.

Option1: Along with a standard Link entity, we can add a Satellite, but that Satellite does not contain the load end date attribute.

Option2: At times, the transaction attributes are added directly to a Link structure, because of which Satellites are not added. This option needs to be avoided because it will completely change the Data Vault model design because the descriptive attributes are added only to a Satellite in the Data Vault model.

According to Linstedt [LO16], IoT data can be modeled as a transactional link as these data are not altered. A Link always has at least two Hubs (maybe same Hub twice), but in the case of IoT data in a transactional link, it would be hard to say what the two hubs would be. This highly depends on the business case and may not always be the best possible solution. Therefore, we present alternative approaches to this problem.

Approach 2.2 [AP 2.2]: IoT data added to a Sensor Hub

The devices that generate the IoT data are modeled as a Hub. As shown in Figure 6.23, *Sensor ID* is the business key for a *Hub_Sensor*. The advantage of modeling the sensor as a Hub is that we can add any number of satellites to it.

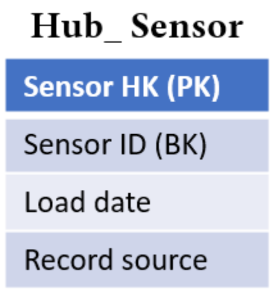


Figure 6.23: Sensor Modeled as a Hub

The temperature sensor data can be stored in a normal Satellite. As shown in Figure 6.24, *Sat_Temp_data* store the attributes like *Value*, *Timestamp*, *Max range*, etc. Here, *Load End date* represents the date and time after which the entries in a Satellite are not valid. In this case, the *Load End date* is filled based on when the entry is added, and as we know, it will expire after a particular period. For instance, a temperature sensor sends a new value every minute, and we capture an entry at timestamp *08:00:00*, then we can set the *Load End date* to *08:00:59*, since we know that there will be a new entry at *08:01:00*. Nevertheless, the main problem in this approach is that we need to update the *Load End date* each time when the new entry comes from the source system, which is not a good idea for the IoT data. Therefore, the *Load End date* is left out for IoT data, as IoT data are only valid when it was captured.

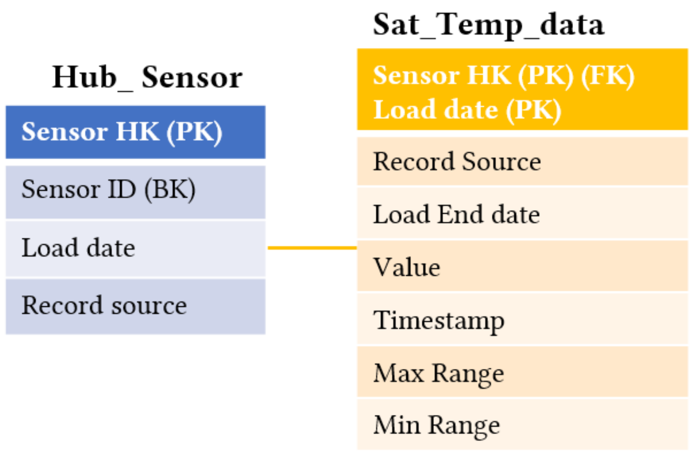


Figure 6.24: IoT Temperature Data Stored in a Satellite

Alternative 1 [AL 1]: IoT data in a Satellite

Based on the idea of Transactional Links along with the business logic, we do not add a Satellite (without a load end date) to a Link but instead, add it to a Hub. Following this, IoT data can be stored in a Satellite without load end date along with a Hub. This approach provides an alternative solution to AP 2.1.

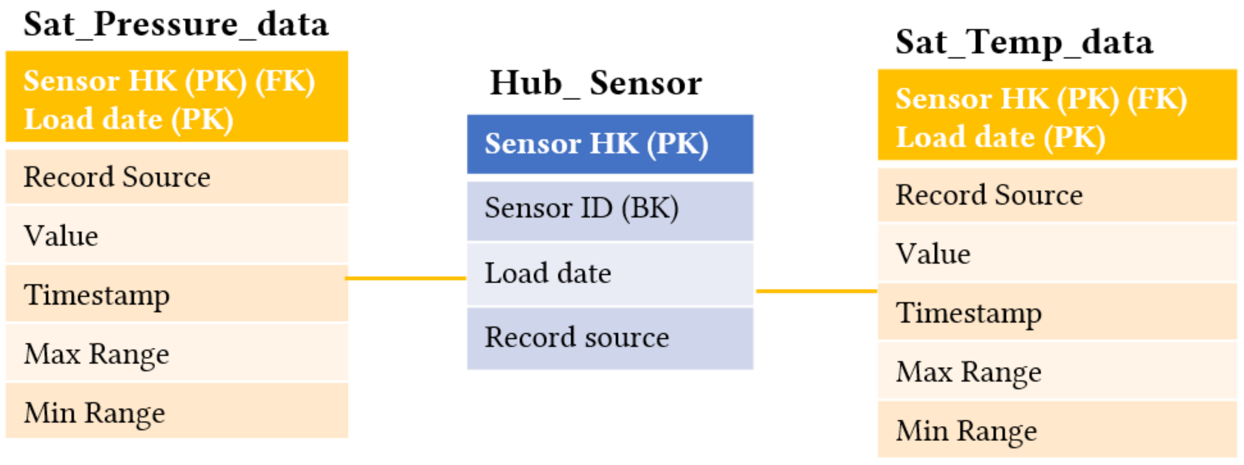


Figure 6.25: Temperature and Pressure Sensor Data Stored in a Special Satellite

As shown in Figure 6.25, *Hub_Sensor* is connected to two satellites namely *Sat_Pressure_data* and *Sat_Temp_data*. These two special satellites store attributes like *Timestamp*, *Max Range* and *Min Range* without Load End Date.

Alternative 2 [AL 2]: Path to an External System

Until now we have seen the approaches that store IoT data directly in a satellite. However, it is also possible to store IoT data as JSON in an external system (HDFS, MongoDB, etc.) and store the path to the actual file in a Satellite. As shown in Figure 6.26, *Hub_Sensor* has two satellites namely *Sat_Temp_data* and *Sat_Pressure_data*. These satellite values are stored as JSON data in an external system, and the path to that file is stored in that Satellite.

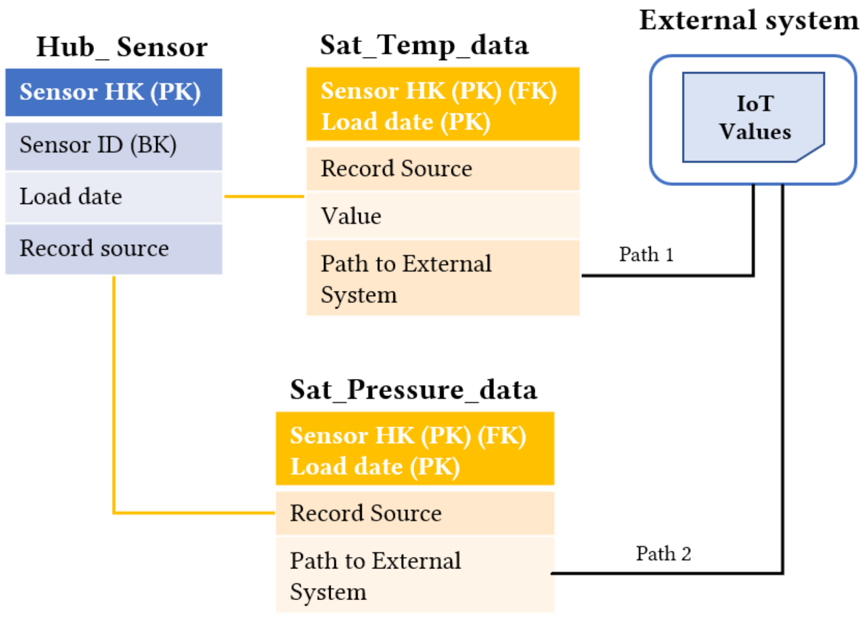


Figure 6.26: IoT Data Stored Using an External Link

Approach 2.3 [AP 2.3]: IoT data added to a Business Object Hub

The business objects like *machine*, *product*, *production process* can be modeled as a Hub. In Figure 6.27, the production process is modeled as a Hub namely *Hub_Production* with *Production ID* and *Line number* as composite business keys.

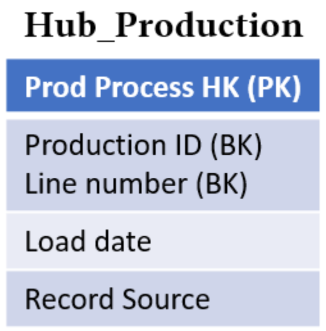


Figure 6.27: Production Process Modeled as a Hub

Alternative 1 [AL 1]: IoT data in a Satellite

Here, the business object is modeled as a *Hub_Production* and is connected to two Satellites that possess IoT data namely *Sat_Pressure_data* and *Sat_Temp_data*. These two special Satellites store attributes like *Timestamp*, *Max Range* and *Min Range* without *Load End Date* as shown in Figure 6.28.

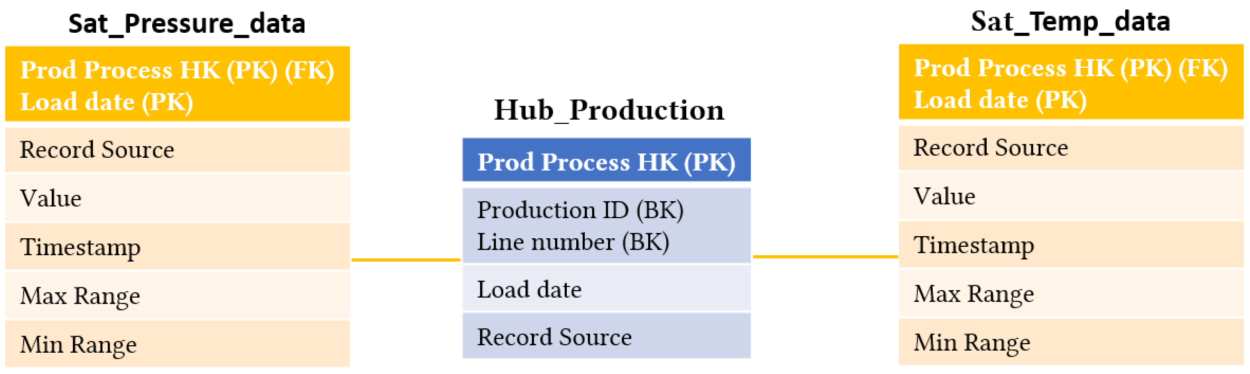


Figure 6.28: Temperature and Pressure Data Added to a Production Process Hub

Alternative 2 [AL 2]: Path to an External System

It is possible to store IoT data as JSON in an external system (HDFS, MongoDB, etc.) and store the path to the actual file in a Satellite. As shown in Figure 6.29, *Hub_Production* has two special Satellites namely *Sat_Temp_data*, and *Sat_Pressure_data* without *Load End Date*. The values of *Sat_Temp_data* and *Sat_Pressure_data* are stored as JSON data in an external system and the path to that file is stored in that Satellite.

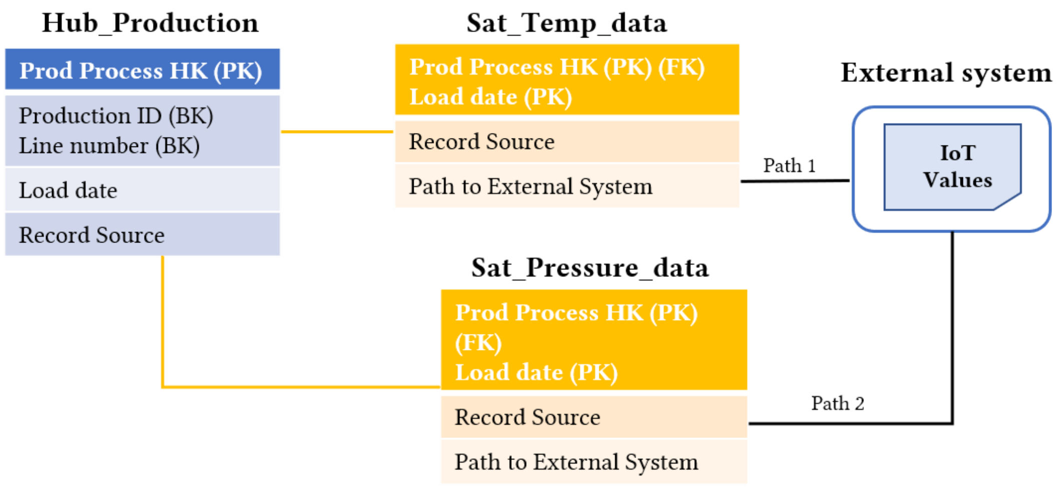


Figure 6.29: IoT Data Stored Using an External Link

Example

Let us use the approaches explained above to model the Data Vault for the use case "Autonomous Car Testing" described in Subsection 3.3. In this use case, we integrate structured and unstructured data into the Data Vault model.

Car, *Test drive* and *Sensor Hub* are identified as core business objects with unique business keys. Hence, they are modeled as Hubs. As shown in Figure 6.30, *Hub_Car* has the business key *Car ID*. Likewise, *Hub_Testdrive* has the business key *Testdrive ID*, and *Hub_Sensor* has the business key *Sensor ID*.

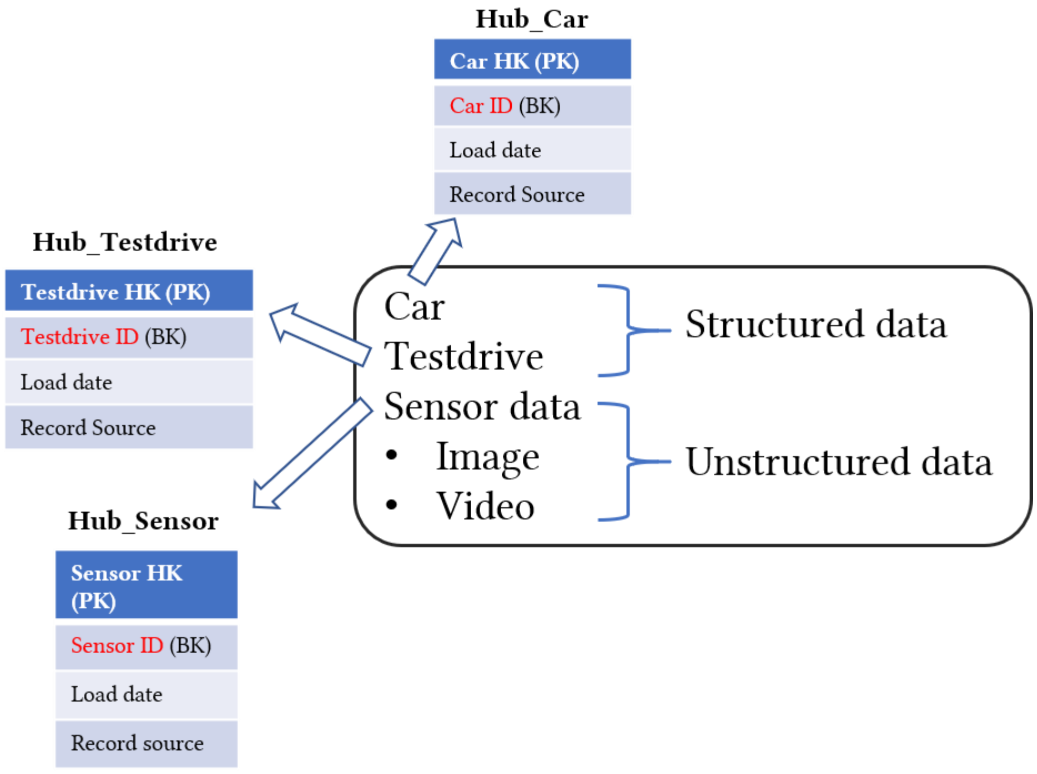


Figure 6.30: Hub Entities in Autonomous Car Testing

Next step is to find the relationship between Hubs. *Link_Car Testdrive* exists between *Hub_Car* and *Hub_Testdrive*. Similarly, *Link_Car Sensor* exists between *Hub_Car* and *Hub_Sensor* as shown in Figure 6.31.

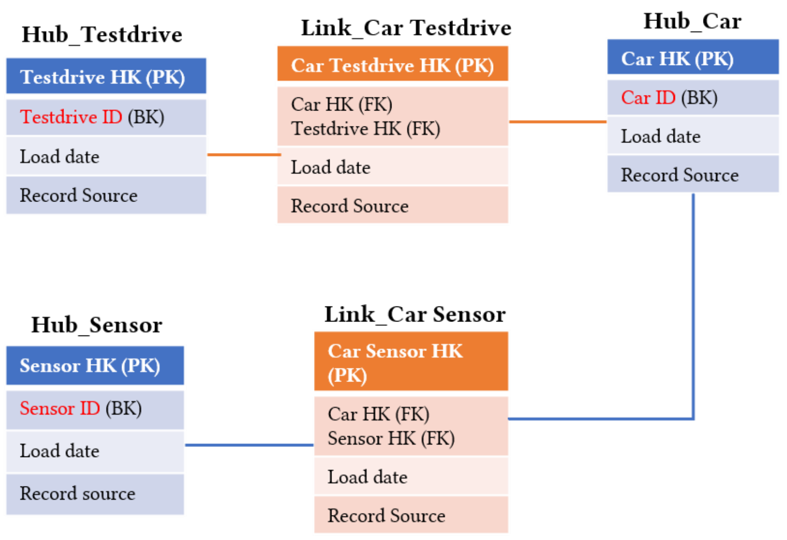


Figure 6.31: Link Entities in Autonomous Car Testing

Next step is to store the descriptive attributes of *Hub_Car* and *Hub_Testdrive* in separate Satellites. The most interesting part is to know how unstructured data like IoT data can be modeled based on the already discussed approaches.

The sensor image data like *Timestamp*, *Image Name*, *Image File* are stored directly in a Satellite *Sat_Image* based on AP 1.2 - AL 1. The video files are stored in an external system, and the location is stored in a Satellite Sat_Video based on AP 1.2 - AL 2 and the IoT temperature data are stored directly in a Satellite *Sat_Temp_data* based on AP 2.2 - AL 1 as shown in Figure 6.32.

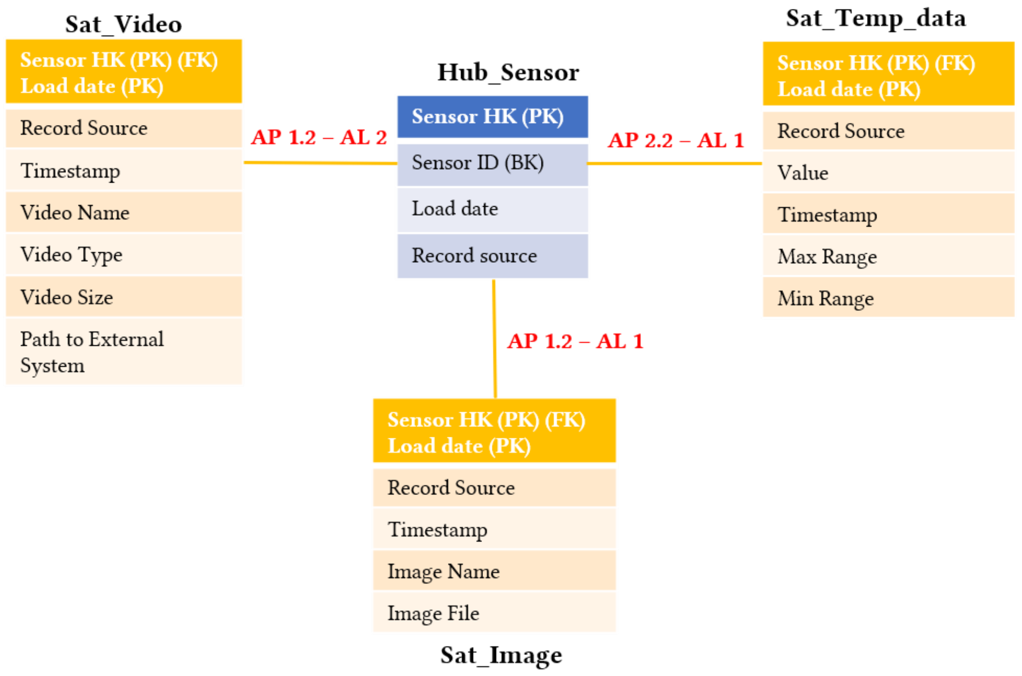


Figure 6.32: IoT Data Modeled as a Satellite

The IoT data generated during the Autonomous Car Testing is enormous. Therefore, it should be decided whether to store the IoT data attributes directly in a Satellite based on AP 2.2 - AL 1 or to store the IoT data in an external system based on AP 2.2 - AL 2, which is illustrated in Figure 6.33.

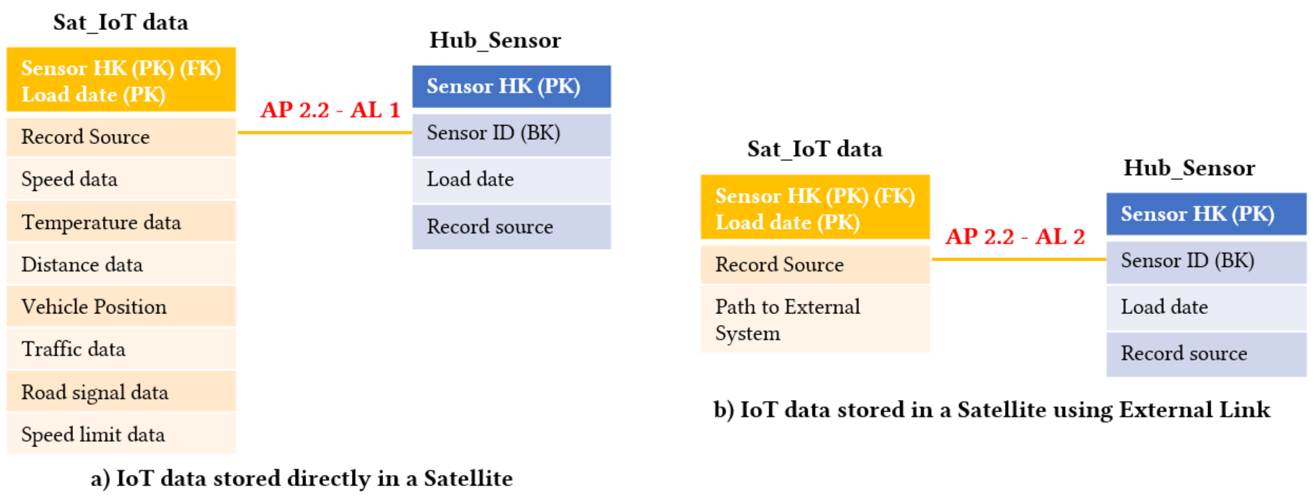


Figure 6.33: IoT Data Modeled as a Satellite Using Two Different Approach

7 Implementation

In this chapter, we will see the prototypical implementation of some of the integration approaches using multiple use cases from different areas like CRM, Manufacturing, and Autonomous Car Testing. Hadoop and Hive are used to implement the use cases.

Hadoop[4] is used to store and process the large volume of data reliably with HDFS and Map Reduce respectively. HDFS supports to store files of any format like images, JSON, audios, videos, etc. Hadoop is chosen to implement our work due to the following reasons:

- It can handle heterogeneous data especially unstructured data.
- It supports hash keys.
- It supports the storage of structured data in table structures via its relational extensions, e.g., Hive and HBase.

Hive[5] is a tool for the data warehouse infrastructure built on top of Hadoop, and it is designed to load and transform heterogeneous data into HDFS. Hive is used to analyze and query the data. It makes use of Hive Query language (HiveQL), which is similar to Structured Query Language (SQL). Hive queries are translated into Map Reduce jobs, which are later submitted to Hadoop cluster. Hive separates the schema and data; schema is stored inside a database and data are stored inside the HDFS. The reasons to choose Hive are:

- It is extensible and scalable.
- It will enable us to write simple queries to perform map reduce jobs.
- Hive integrates well with HDFS.
- Data can be loaded directly into HDFS or in a Hive table.

For various use cases, the Data Vault tables like Hubs, Links and Satellites are created with different fields based on the suitable approaches and metadata elements of the Data Vault entities as shown in the Tables 7.1 and 7.2 (For simplicity, only the tables, which are necessary to implement different approaches are taken into consideration). As discussed earlier in Subsection 6.2, there are different approaches to integrate unstructured data into the Data Vault model. Among these, some of the approaches are implemented here.

To implement AP 1.1 - AL 2

Here, CRM is used as an example area, though the approach would be applicable to any area. For instance, a customer ordered a product and then sent a complaint email that contains an image. This approach is primarily for unstructured data like image/audio/video that has the business key. In this case, the image has a composite business key *CID_PID_ImID* and hence the image is modeled as a Hub table *Hub_image* along with its descriptive attributes stored in a Satellite table *S_image*. The mandatory attributes of satellites like hash key *image_HK*, *loaddate* and along with the other attributes *date_time*, *image_name* are stored directly in a Satellite. However, the image file attribute is stored in HDFS, and the actual path is stored in the *S_image* table with the attribute name *path_to_HDFS*. The tables are created and data are loaded into them from Comma

4 https://hadoop.apache.org/
5 https://cwiki.apache.org/confluence/display/Hive/

Separated Values (CSV) files. Figure 7.1 shows the queries to create the tables and add the data to HDFS.

Table name	Field names with its data types	Approach - Alternative
Hub_Customer	**customer_HK** string, **customer_id** int, **loaddate** timestamp, **recordsource** string	
Sat_Customer	**customer_HK** string, **loaddate** timestamp, **loadenddate** timestamp, **recordsource** string, **name** string, **address** string, **phone_no** int, **email** string	
Hub_Product	**product_HK** string, **product_id** string, **loaddate** timestamp, **recordsource** string	
Sat_Product	**product_HK** string, **loaddate** timestamp, **loadenddate** timestamp, **recordsource** string, **name** string, **prod_type** string, **price** int	
Hub_Image	**image_HK** string, **CID_PID_ImID** string, **loaddate** timestamp, **recordsource** string	AP 1.1 - AL 2
S_Image	**image_HK** string, **loaddate** timestamp, **loadenddate** timestamp, **recordsource** string, **date_time** timestamp, **image_name** string, **path_to_HDFS** string	
Hub_Email	**email_HK** string, **email_id** string, **loaddate** timestamp, **recordsource** string	AP 1.2 - AL 2
Sat_Email	**email_HK** string, **loaddate** timestamp, **loadenddate** timestamp, **recordsource** string, **date_time** timestamp, **toadd** string, **fromadd** string, **subject** string	
S_E_Image	**email_HK** string, **loaddate** timestamp, **loadenddate** timestamp, **recordsource** string, **date_time** timestamp, **image_name** string, **path_to_HDFS** string	

Table 7.1: Data Vault Tables for CRM

To implement AP 1.2 - AL 2

This approach is primarily for unstructured data like image/audio/video that do not have the business key on its own. Using the same CRM scenario as above, the image is modeled as a *Sat_Image*, and it is linked to a Hub it belongs to, i.e. *Hub_Email*. However, the image file is stored externally in HDFS, and the actual path is stored in the *Sat_Image* table with the field name *path_to_HDFS*. The tables are created and data are loaded into them from CSV files.

To implement AP 2.2 - AL 1

Here, Manufacturing is used as an example area, though the approach would be applicable to any area. For instance, we assume a machine with a temperature and pressure sensor that sends values are used to monitor the performance of the machine. This approach is mainly for the unstructured IoT data, which stores these data directly in a Satellite table. The sensor with the business key is modeled as a Hub *Hub_Sensor* and the IoT values like pressure and temperature data with attributes *datetime*, *maxvalue* and *minvalue* are directly stored in the Satellites namely

S_Temperature and *S_Pressure*. The tables are created and data are loaded into them from CSV files.

To implement AP 2.2 - AL 2

This approach is mainly for the unstructured IoT data, which are stored in an external system. Using the same CRM scenario as above, the sensor with the business key is modeled as a Hub *Hub_Sensor*. The IoT values like pressure and temperature data with attributes *datetime*, *maxvalue* and *minvalue* are stored in HDFS, and the actual path is stored in the Satellites namely *S_Temperature* and *S_Pressure*. The tables are created and data loaded into them from CSV files.

```
//To create a Hive table
hive> create Hub_Image (image_HK string, CID_PID_ImID string, loaddate timestamp, recordsource
string)
    > row format delimited
    > fields terminated by ','
    > stored as textfile;

//To load the data from local file system into Hive table
hive> load data local inpath '/home/cloudera/Documents/hive_table/H_image.csv' into table Hub_Image;

//To create directory in Hadoop
[cloudera@quickstart Downloads]$ hadoop fs -mkdir /CRM

//To load the data from local file system into HDFS
[cloudera@quickstart ~]$ hadoop fs -put /home/cloudera/Documents/hive_table/Image1.jpg CRM
```

Figure 7.1: Sample Queries in Hive and Hadoop

Table name	Field names with its data types	Approach - Alternative
Hub_Sensor Sat_Temperature Sat_Pressure	**sensor_HK** string, **sensor_id** string, **loaddate** timestamp, **recordsource** string **sensor_HK** string, **loaddate** timestamp, **recordsource** string, **date_time** timestamp, **value** int, **maxvalue** int, **minvalue** int **sensor_HK** string, **loaddate** timestamp, **recordsource** string, **date_time** timestamp, **value** int, **maxvalue** int, **minvalue** int	AP 2.2 - AL 1
Hub_Sensor S_Temperature S_Pressure	**sensor_HK** string, **sensor_id** string, **loaddate** timestamp, **recordsource** string **sensor_HK** string, **loaddate** timestamp, **recordsource** string, **path_to_HDFS** string **sensor_HK** string, **loaddate** timestamp, **recordsource** string, **path_to_HDFS** string	AP 2.2 - AL 2
Hub_Production Sat_Temperature Sat_Pressure	**prod process_HK** string, **production_id** string, **line_number** int, **loaddate** timestamp, **recordsource** string **prod process_HK** string, **loaddate** timestamp, **recordsource** string, **date_time** timestamp, **value** int, **maxvalue** int, **minvalue** int **prod process_HK** string, **loaddate** timestamp, **recordsource** string, **date_time** timestamp, **value** int, **maxvalue** int, **minvalue** int	AP 2.3 - AL 1
Hub_Production S_Temperature S_Pressure	**prod process_HK** string, **production_id** string, **line_number** int, **loaddate** timestamp, **recordsource** string **prod process_HK** string, **loaddate** timestamp, **recordsource** string, **path_to_HDFS** string **prod process_HK** string, **loaddate** timestamp, **recordsource** string, **path_to_HDFS** string	AP 2.3– AL 2

Table 7.2: Data Vault Tables for Manufacturing

Sample Queries

These are some of the sample queries to retrieve the data that are implemented based on the various approaches with its alternatives as shown in Figure 7.2.

a) To retrieve the image name and path of the image in a Satellite for a particular image id '*C100_P100_Im100*' based on **AP 1.1 - AL 2**.

b) To retrieve the image name and path of the image in a Satellite for a particular email_id '*E3211*' based on **AP 1.2 - AL 2**.

c) To retrieve the sensor_id, value of the IoT pressure data whose value are above the max value or below the min value based on **AP 1.1 - AL 1**. It helps to identify the machines, which are not in the normal state.
d) To retrieve the sensor_id and the path of the IoT pressure data in a Satellite with the help of sensor_id '*S102*' based on **AP 2.2 - AL 2**.
e) To retrieve the sensor_id, the value of the IoT pressure and temperature data whose value are above the max value or below the min value based on **AP 1.1 - AL 1**. It helps to identify the machines, which are not in the normal state.

```
//Query (a): AP 1.1 - AL 2
hive> select S.image_name, S.path_to_HDFS from S_Image as S
   > inner join Hub_Image as H on H.image_HK=S.image_HK
   > where H.CID_PID_ImID='C100_P100_Im100';

//Query (b): AP 1.2 - AL 2
hive> select S.image_name, S.path_to_HDFS from S_Image as S
   > inner join Hub_Email as H on H.email_HK=S.email_HK
   > where H.email_id='E3211';

//Query (c): AP 2.2 - AL 1
hive> select H.sensor_id, SP.value  from Hub_Sensor as H
   > inner join Sat_Pressure as SP on H.sensor_HK=SP.sensor_HK
   > where SP.value>SP.maxvalue or SP.value<SP.minvalue ;

//Query (d): AP 2.2 - AL 2
hive> select H.sensor_id, SP.path_to_HDFS from Hub_Sensor as H
   > inner join S_Pressure as SP on H.sensor_HK=SP.sensor_HK
   > where H.sensor_id='S102';

//Query (e): AP 2.2 - AL 1
hive> select H.sensor_id, SP.value, ST.value from Hub_Sensor as H
   > inner join Sat_Pressure as SP on H.sensor_HK=SP.sensor_HK
   > where SP.value>SP.maxvalue or SP.value<SP.minvalue
   > union all select H.sensor_id, ST.value from Hub_Sensor as H
   > inner join Sat_Temperature as ST on H.sensor_HK=ST.sensor_HK
   > where ST.value>ST.maxvalue or ST.value<ST.minvalue;
```

Figure 7.2: Sample Queries to Retrieve the Data Implemented Based on Various Approaches

Hence from the prototypical implementation, it provides a proof of concept that with the help of these approaches we can integrate unstructured data into the Data Vault model.

8 Evaluation

In this chapter, we will discuss the technical evaluation (in Subsection 8.1) and theoretical evaluation (in Subsection 8.2) by comparing the various approaches with its alternatives.

8.1 Technical Evaluation

Storage Space

The term storage space refers to how much memory each table takes to store the data in it. Here, a sensor Hub table, temperature Satellite tables, and pressure Satellite tables, which are needed to perform the approach AP 2.2 are considered.

All the tables consist of 100 entries and the size of a sensor Hub table that stores the sensor data are 5092 bytes. The size of a table that stores the temperature data and pressure data directly in the Satellites based on AP 2.2 - AL 1 are 7492 and 7792 bytes respectively. Likewise, the size of a table that stores the temperature data and pressure data in HDFS with the actual path stored in the Satellites based on AP 2.2 - AL 2 are 9392 and 9092 bytes.

The Satellite tables that store the path consumes more memory because for each entry a path is added in a Satellite table. However, the benefit of the adding path would be that one path could reference one document that contains hundreds or thousands of measurement values. Therefore, in order to massively reduce the storage, the alternative idea is to store a path that reference a document.

Time Complexity

Time complexity refers to how much time it takes to produce the result for each query. To evaluate the time taken to retrieve the data based on AP 2.2 - AL 1 and AP 2.2 - AL 2, we have considered the queries (c) and (d), which are implemented in Section 7. The Satellite tables consist of 100 datasets, which are queried 20 times and the time taken to retrieve the data are noted down as shown in Table 8.1.

For query (c), the average time to retrieve the IoT data stored directly in a Satellite (AP 2.2 - AL 1) is 88.3569 seconds and for the query (d) the average time to retrieve the path stored in a Satellite (AP 2.2 - AL 2) is 89.38025 seconds. From this evaluation, it is evident that AP 2.2 - AL 1 takes less time to retrieve the data when compared to AP 2.2 - AL 2. Furthermore, based on AP 2.2 - AL 2, if we want to retrieve the data, additionally we need to go to HDFS and retrieve the data, which would take a longer time than this.

Query Complexity

From the database user's perspective, query complexity is measured with the help of various metrics like [VJ16]:

- Number of tables used in a query
- Number of columns in a query
- Length of a query

- Number of operators like Joins, Scans in a query
- The number of expression operators involved in the query like less than, greater than, equal to, OR, AND etc.

Time taken to retrieve the IoT data stored in a Satellite (seconds)	Time taken to retrieve the IoT data stored in HDFS and the actual path stored in a Satellite (seconds)
92.847	84.481
93.874	84.861
80.313	91.652
81.444	89.716
86.759	86.152
94.481	91.067
96.891	88.96
87.816	90.685
92.052	87.695
89.538	89.243
84.297	88.5
87.985	91.163
89.159	91.72
83.924	92.094
89.752	91.739
91.312	89.637
85.461	89.891
87.024	87.107
87.05	91.428
85.159	89.814
88.3569	**89.38025**

Table 8.1: Time Complexity of AP 2.2 - AL 1 and AP 2.2 - AL 2

Taking all these metrics into consideration, the query is said to be more complex when the query involves more tables, referenced columns, operators, expression operators, length and runtime. To measure the query complexity of AP 2.2, we have considered the queries (c), (d) and (e), which are implemented in Section 7.

Query (c) based on AP 2.2 - AL 1 involves two tables namely sensor Hub table and pressure Satellite table, one join, one referenced column and three expression operators '>', '<', 'OR'.

Query (d) based on AP 2.2 - AL 2 involves two tables namely sensor Hub table and pressure Satellite table, one join, one referenced column, and one expression operator ‘=’.

Query (e) based on AP 2.2 - AL 1 involves three tables namely sensor Hub table, pressure Satellite table, and temperature Satellite table, two joins, two referenced columns and three expression operators ‘>’, ‘<’, ‘OR’ used twice.

On evaluation, we notice that the query complexity of query (c) and (d) are both equally complex. However, for query (d) we also have to retrieve the data from HDFS, which adds complexity. Therefore, AL 1 is less complex.

8.2 Theoretical Evaluation

Advantages and Disadvantages of Various Approaches

The developed approaches described in Subsection 6.2, to integrate unstructured data into the Data Vault model are compared to the rules for modeling the Data Vault entities discussed in the Subsection 4.1.4. This comparison helps to determine whether the developed approaches conform to the modeling rules of Data Vault. From this comparison, it is clear that all these approaches (AP 1.1, AP 1.2, AP 2.2, AP 2.3) with their alternatives (AL 1 and AL 2) are modeled adhering to the rules.

Below, we have compared the various approaches to list out the advantages and disadvantages of each approach.

AP 1.1: Unstructured data like image/audio/video are modeled as a Hub.

+ A Hub can be linked to other Hubs easily with the help of a Link, which provides flexibility.
+ We can easily add more Satellites to a Hub, which provides scalability.
- It is not possible to use this approach if unstructured data does not have the business key.

AP 2.2: Unstructured IoT data added to a Satellite.

+ IoT data added to a Satellite without the load end date do not need an additional effort of updating the load end date, and it is a suitable approach for the IoT data as the data are valid only for that particular time.
- IoT data can be added to a normal Satellite, but it needs an additional effort to update the load end date whenever a new value arrives.

AL 1: File in a Satellite

+ Time taken to retrieve the data are faster when compared to AL 2, the reason is that the data can be directly accessed.
- As the data are directly stored in a Satellite table, it is not flexible to add the data from other tables easily. Also, the Satellites will become big rather quickly.

AL 2: Path to an external system

+ If there are lots of measurements, one path can reference a document that contains a few hundreds or thousands of references. This reduces the table storage space.
- Performance is less when compared to AL 1, which are measured based on time complexity. The reason is that to retrieve the data from a Satellite table; it involves a two-step process, i.e. from a Satellite table and then from HDFS.

From these comparisons, it is evident that if we want to retrieve the data faster irrespective of the memory consumption of the table, then it is a good option to use AP 2.2 - AL 1. However, if we want to store a large amount of data with less memory consumption irrespective of the performance, then it is a good option to use AP 2.2 - AL 2.

9 Summary and Future Work

Hubs, Links, and Satellites form the basic entities of the Data Vault model. Similar to the traditional modeling techniques like Kimball's approach and Inmon's approach, the Data Vault model is mainly focused on modeling structured data. In recent days, the data produced by the real-world use cases such as in the fields of CRM, Manufacturing and Autonomous Car Testing are heterogeneous. In such cases, there are no well-defined approaches to integrate unstructured and semi-structured data into the Data Vault model.

There exist few approaches to integrate semi-structured data into the Data Vault model. However, there are no approaches to integrate unstructured data into the Data Vault model. Therefore, in this thesis, we have developed various approaches to integrate unstructured data into the Data Vault model using examples from CRM, Manufacturing and Autonomous Car Testing. For both unstructured image/audio/video data and IoT data, two alternative approaches were proposed and discussed. Data in these alternatives are either stored directly in the Data Vault model, or stored in an external system and accessed via links. We have implemented some use cases from CRM, and Manufacturing for the various approaches along with their two alternatives.

For evaluation, we have compared the advantages and disadvantages of these approaches and also discussed the storage space, time complexity, and query complexity to retrieve the IoT data using AP 2.2 - AL 1 and AP 2.2 - AL 2. The choice of the correct alternative depends on the use case. Thus, using the approaches defined in this thesis, we can integrate the unstructured and semi-structured data along with the structured data into the Data Vault model by satisfying the Data Vault characteristics.

Future Work

The approaches discussed in this thesis to integrate unstructured and semi-structured data into the Data Vault model can be used in the future to validate it with other use cases. Another idea is to identify if there is an alternative way to store unstructured data using an external system. In this thesis, we have measured the performance of the approaches to retrieve the IoT data. In the future, it would be interesting to implement the approaches to store image/video/audio data directly in a Satellite table and measure the performance by comparing it with the image/video/audio data that are stored in an external system. Also, it would be interesting to see how much longer it takes actually to retrieve the data from an external system and how exactly storing the data externally affects storage space.

References

[B04] M. Breslin, “Data Warehousing Battle of the Giants: Comparing the Basics of the Kimball and Inmon Models”, Business Intelligence Journal, 2004.

[BB13] B. Boyina, T. Breur, “Adapting Data Warehouse Architecture to Benefit from Agile Methodologies”, 2013.

[BFG+06] C. Ballard, D. M. Farrell, A. Gupta, C. Mazuela, S. Vohnik “Dimensional Modeling: in a Business Intelligence Environment”, International Technical Support Organization, Redbooks, IBM, 2006.

[BGM+11] J. Boyer, S. Gao, S. Malaika, M. Maximilien, R. Salz, J. Simeon, “Experiences with JSON and XML Transformations”, IBM Submission to W3C Workshop on Data and Services Integration, 2011.

[BH15] E. S. Btoush, M. M. Hammad, “Generating ER Diagrams from Requirement Specifications based on Natural Language Processing”, international Journal of Database Theory and Application, Vol.8, No.2, pp.61-70, 2015.

[C76] P. P. Chen, “The Entity-Relationship Model - Toward a Unified View of Data", ACM Transactions on Database Systems, Volume. 1, No. 1, 1976.

[CGP16] A. Caputo, G. Marzi, M. Pellegrini, “The Internet of Things in Manufacturing Innovation Processes: Development and Application of a Conceptual Framework”, Business Process Management Journal, Volume. 22, No. 2, 2016.

[CJJ18] K. Cernjeka, D. Jaksic, V. Jovanovic, “NoSQL Document Store Translation to Data Vault Based EDW”, MIPRO 2018.

[DWL+00] G. Dobbie, X. Wu, T. W. Ling, M. L. Lee, “ORA-SS: An Object-Relationship-Attribute Model for Semi-Structured Data”, The National University of Singapore, 2000.

[F14] A. E. Fentaw, “Data Vault Modeling: An Introductory Guide”, Helsinki Metropolia University of Applied Sciences, Thesis, 2014.

[GL11] K. Graziano, D. Linstedt, “Introduction to Data Vault Modeling”, 2011.

[GM16] K. Gandhi, N. Madia, “Information Extraction from Unstructured Data Using RDF”, International Conference on ICT in Business Industry & Government, 2016.

[GSM14] C. Gröger, H. Schwarz, B. Mitschang, “The Deep Data Warehouse: Link-Based Integration and Enrichment of Warehouse Data and Unstructured Content”, Proceedings of the 2014 IEEE 18th International Enterprise Distributed Object Computing Conference, 2014.

[H02] W. H. Inmon, “Building the Data Warehouse”, Wiley, third edition, 2002.

[H12] H. Hultgren, “Data Vault Modeling Guide: Introductory Guide to Data Vault Modeling”, Genesee Academy, LLC, 2012.

[KJ13] C. Knowles, V. Jovanovic, “Extensible Markup Language (XML) Schemas for Data Vault Models, Journal of Computer Information Systems, 2013.

[KJ13] C. Knowles, V. Jovanovic, “Extensible Markup Language (XML) Schemas for Data Vault Models”, Journal of Computer Information Systems, 2013.

[KR13] R. Kimball, M. Ross, “The Data Warehouse ToolKit: The Definitive Guide to Dimensional Modeling”, Wiley, Third Edition, 2013.

[L18] D. Linstedt, “Data Vault Modeling Specification V2.0.2, Focused on the Data Model Components”, 2018.

[Lin14] D. Linstedt, “Data Vault 2.0 Hashes Verses Natural Keys”, 2014 [Online] https://danlinstedt.com/allposts/datavaultcat/datavault-2-0-hashes-versus-natural-keys/

[Lin17] D. Linstedt, “Data Vault 2.0, Hashes, one more time”, 2017 [Online] https://danlinstedt.com/allposts/datavaultcat/datavault-2-0-hashes-one-more-time/

[LKL+15] Y. H. Liu, J. F. Kung, J. Lin, Y. B. Hsu, “Using Text Mining to Handle Unstructured Data in Semiconductor Manufacturing”, International Symposium on Semiconductor manufacturing, IEEE, 2015.

[LLL+99] S. Y. Lee, M. L. Lee, T.W. Ling, L. A. Kalinichenko, “Designing Good Semi-Structured Databases” International Conference on Conceptual Modeling, pp. 131-145, 1999.

[LLY+11] X. Liu, B. Lang, W. Yu, J. Luo, L. Huang “AUDR: An Advanced Unstructured Data Repository”, IEEE, 2011.

[LO16] D. Linstedt, M. Olschimke, “Building a Scalable Data Warehouse with Data Vault 2.0”, Elsevier, 2016.

[LWD+05] Ling, T. Wang, Dobbie, Gill, “Semi-Structured Database Design - Data Models for Semi-Structured Data”, Chapter. 2, Springer, 2005.

[MT16] N. G. Miloslavskaya, A. Tolstoy, “Big Data, Fast Data and Data Lake Concepts”, Elsevier, Volume 88, BICA 2016.

[N13] F. Nogatz, “From XML Schema to JSON Schema Comparison and Translation with Constraint Handling Rules”, 2013.

[NJ16] Z. Naamane, V. Jovanovic, “Effectiveness of Data Vault Compared to Dimensional Data Marts on Overall Performance of a Data Warehouse System”, International Journal of Computer Science Issues, Volume. 13, No. 4, 2016.

[P14] M. Y. Patil, “Social Media and Customer Relationship Management”, IOSR Journal of Business and Management, P-ISSN: 2319-7668, pp. 27-32, 2014.

[PG12] D. Pilev, A. Georgieva, “Effective Time Temporal Database Model”, International Journal on Information Technologies & Security, No. 2, 2012.

[PP12] A. R. Patel, J. M. Patel, “Data Modeling Techniques for Data Warehouse”, International Journal of Multidisciplinary Research, Volume. 2, No. 2, 2012.

[QS13] Q. K. Quboa, M. Saraee, “A State-of-the-Art Survey on Semantic Web Mining”, Intelligent Information Management, Volume. 5, pp. 10-17, 2013.

[S14] P. Stiglich. “Data Modeling in the Age of Big Data”, Business Intelligence Journal, Volume. 19, No. 4, pp. 17–22, 2014.

[Tha17] R. Thakur, “Infrared for Autonomous Vehicles”, Recent Development in Optoelectronic Devices, 2017. http://dx.doi.org/10.5772/intechopen.70577

[VJ16] A. Vashistha, S. Jain, “Measuring Query Complexity in SQLShare Workload”, Proceedings of the International Conference on Management of Data, 2016.

[VNR14] N. Veeranjaneyulu, M. Nirupama Bhat, A. Raghunath, “Approaches for Managing and Analyzing Unstructured Data”, International Journal on Computer Science and Engineering, Volume. 6, 2014.

[WLL+02] X. Wu, T. W. Ling, M. L. Lee, G. Dobbie, “Designing Semi-Structured Databases using ORA-SS Model“, Proceedings of the Second International Conference on Web Information Systems Engineering, IEEE, 2002.

[YL16] L. Yessad, A. Labiod, “Comparative Study of Data Warehouses Modeling Approaches: Inmon, Kimball and Data Vault”, International Conference on System Reliability and Science, 2016.

Declaration

I hereby declare that the work presented in this thesis is entirely my own. I did not use any other sources and references that the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations. Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted copies.

____
Place, Date, Signature