Fall 2015

Testing Data Vault-Based Data Warehouse

Connard N. Williams

TESTING DATA VAULT-BASED DATA WAREHOUSE

by

CONNARD N. WILLIAMS

(Under the Direction of Vladan Jovanovic)

ABSTRACT

Data warehouse (DW) projects are undertakings that require integration of disparate sources of data, a well-defined mapping of the source data to the reconciled data, and effective Extract, Transform, and Load (ETL) processes. Owing to the complexity of data warehouse projects, great emphasis must be placed on an agile-based approach with properly developed and executed test plans throughout the various stages of designing, developing, and implementing the data warehouse to mitigate against budget overruns, missed deadlines, low customer satisfaction, and outright project failures. Yet, there are often attempts to test the data warehouse exactly like traditional back-end databases and legacy applications, or to downplay the role of quality assurance (QA) and testing, which only serve to fuel the frustration and mistrust of data warehouse and business intelligence (BI) systems. In spite of this, there are a number of steps that can be taken to ensure DW/BI solutions are successful, highly trusted, and stable. In particular, adopting a Data Vault (DV)-based Enterprise Data Warehouse (EDW) can simplify and enhance various aspects of testing, and curtail delays common in non-DV based DW projects. A major area of focus in this research is raw DV loads from source systems, keeping transformations to a minimum in the ETL process which loads the DV from the source. Certain load errors, classified as permissible errors and enforced by business rules, are kept in the Data Vault until correct values are supplied. Major transformation activities are pushed further downstream to the next ETL process which loads and refreshes the Data Mart (DM) from the Data Vault.

INDEX WORDS**:** Data warehouse testing, Data Vault, Data Mart, Business Intelligence, Quality Assurance, Raw Data Vault loads

TESTING DATA VAULT-BASED DATA WAREHOUSE

by

CONNARD N. WILLIAMS

B.Sc., University of the West Indies, Jamaica, 1998

A Thesis Submitted to the Graduate Faculty of Georgia Southern University in Partial Fulfillment of the Requirements for the Degree

MASTER OF SCIENCE

STATESBORO, GEORGIA

TESTING DATA VAULT-BASED DATA WAREHOUSE

by

CONNARD N. WILLIAMS

Major Professor:	Vladan Jovanovic
Committee:	Wen-Ran Zhang
	James Harris

Electronic Version Approved:
Fall 2015

DEDICATION

For my wife, Denise
daughter, Sharese
and sons, Daniel and Nate

ACKNOWLEDGEMENTS

I would like to thank my thesis chair Dr. Vladan Jovanovic for the guidance, oversight, and timely responses that he provided during my studies in the MSCS program at Georgia Southern University. I would also like to thank thesis committee members Dr. James Harris and Dr. Wen-Ran Zhang, and all my MSCS professors for the challenges and knowledge shared through the delivery of their course content. Last but not least, I would like to thank my family for their unwavering love and support during my program of study, and especially for putting up with my long hours and late nights doing school work. I could not have done this without you.

TABLE OF CONTENTS

CHAPTER 1
INTRODUCTION

1.1 Overview

A number of factors such as increasing business mergers, data center migrations, and greater focus of upper-level management on data-driven decisions are fueling the need for more effective data warehouse testing. As clearly stated in (Golfarelli and Rizzi, 2009b), testing is an essential part of the design life-cycle of any software product. Moreover, testing is especially critical to success in data warehouse projects because users need to trust in the quality of the information they access.

While some of the testing strategies commonly employed for traditional IT projects are applicable to data warehouse projects, the complexity and scope of data warehouse systems often demand a different approach. Most of the BI/DW systems are like black boxes to customers who primarily value the output reports/charts/trends/KPIs, often overlooking the complex and hidden logic applied behind the scenes (Kamal and Nakul, 2010). The work in this thesis focuses on testing the DV-based data warehouse, highlighting strategies, implications, and impact. Since there are a number of challenges and recommendations for successful implementation that are applicable to both the traditional DW projects and DV-based DW projects, when these are presented they apply to both types of projects. However, the DV-based methodology demands separate investigation for demonstrating its modeling and architectural impact on the testing strategy; this particular aspect is presented in details in the later chapters.

1.2 Structure of this work

Chapter 2 provides background and important definitions in the literature, which create the backdrop against which the majority of this work is aligned. Chapter 3 discusses some architectural considerations and highlights where the DV sits in the DV-based DW architecture. In chapter 4, we briefly look at the implications for the QA team, and present a comparative overview of traditional database testing versus data warehouse testing. In chapter 5 we take a look at a general DW testing. The concepts here are general enough to apply to both traditional DW projects and DV-based projects. Chapter 6 looks at DW testing challenges, and some general recommendations and best practices. While not all the recommendations are applicable

to the DV, most of them are valid and warrant at least an assessment. In chapter 7 we introduce several data models, as well as the load patterns for both DV 1.0 and DV 2.0 that were used in this research. The source data models used represent meaningful example of realistic source (though simplified for greater focus) for integration into raw DV, with subsequent transformations downstream feeding data marts. We also look at an interesting aspect in DV 2.0: the use of hashing instead of sequences for surrogate keys (used for joining), and the impact on ETL workflow and parallel loads. In chapter 8 we look at a number of experiments (using the data models and load patterns introduced earlier in chapter 7) for carrying out raw DV loads and highlights the interesting and novel concept of DV load data errors included and kept on record – no fixes allowed, but rather adding correct values once they are figured out and made available. Chapter 9 presents the conclusion.

CHAPTER 2
BACKGROUND AND DEFINITIONS

2.1 Data Warehousing

According to Golfarelli and Rizzi in (2009a), data warehousing is a collection of methods, techniques, and tools used to support knowledge workers - senior managers, directors, managers, and analyst - to conduct data analyses that help with performing decision-making processes and improving information resources.

2.2 Data Warehouse

A data warehouse is a composite and collaborated data model that captures the entire data of an organization. It brings together data from heterogeneous sources into one single destination. It is not just bringing data. Data is Extracted, Transformed and Loaded (ETL) into the data warehouse. This processing of data is usually done in what is known as a 'staging area' (Mathen, 2010). There is no strict requirement for normalization of the data, because unlike operational databases which are driven by inserts, updates and deletes, the data warehouse data is primarily read-only and is retrieved for querying and analytical purposes. As stipulated by Inmon and outlined in (Vucevic and Zhang, 2011), the major characteristics of the DW are:

- it is subject oriented – provides information about particular subjects
- it is integrated and consistent – the data in the DW is gathered from various, possibly heterogeneous, sources and coherently merged
- it shows its evolution over time and it is not volatile – all data in the DW is identified with a time period, and data is never destroyed

2.3 Data Mart

A data mart (DM) is a subset of the data warehouse data. Data stored in a DM can theoretically be of the same granularity as the DW, but typically it is aggregated to some extent, and includes information relevant for a specific business area, specific department, or specific set of users.

2.4 Business Intelligence

Business Intelligence (BI) application loosely refers to the range of capabilities provided to business users to leverage the presentation area for analytic decision making. By definition, all

BI applications query the data in the DW/BI presentation area. Querying, obviously, is the whole point of using data for improved decision making (Kimball and Ross, 2013).

2.5 Data Vault Modeling

Data Vault modeling was developed by Dan Linstedt and serves to structure the data warehouse data as systems of permanent records, and to absorb structural changes without requiring any data alterations; as outlined in (Jovanovic and Bojicic, 2012; Collins, et al., 2014), these are characteristics which separates it from other modeling approaches.

2.6 Data Vault – EDW

It is worth noting that some studies (Ivanova, et al., 2012; Ivanova, et al., 2013; Martinez-Rubi, et al. 2014) explored database-attached file repositories, where the data is kept in original format and accessed via an array-based query language, as a data vault for processing scientific data files, including Geographic Information System (GIS) and sensor related data. However, for the purpose of this work the data vault, as the EDW, is explored (with definition and treatment) within the context of the model developed by Dan Linstedt.

The EDW, or core historical data repository, consists of the Data Vault modeled tables. The EDW holds data over time at a granular level (raw data sets). The Data Vault (Linstedt and Graziano, 2011; Graziano, 2011; Linstedt and Olschimke, 2015) is comprised of Hubs, Links, and Satellites and they are described as follows:

2.6.1 Hub

The job of a Hub is to store nothing more than a core business concept. It is responsible for tracking the first time the data vault encounters a business key in the warehousing load, and where it came from. Hubs can never contain foreign keys, but store the business key, a uniquely generated sequence id, the timestamp when the data was loaded, and the source of the data. It contains no system attributes for versioning, i.e. no valid to/from dates.

2.6.2 Link

A Link is an intersection of business keys. As such, it defines a relationship between business concepts. Links provide the flexibility to the data vault model by allowing change to the structure over time. Links consist of the uniquely generated sequence ids from the hubs, a

warehouse sequence id (required only if the link will be associated with link satellite), the timestamp when the data was loaded, and the source of the data. Like the hub, it contains no system attributes for versioning.

2.6.3 Satellite

Satellites are the warehousing portion of the data vault. The purpose of the satellite is to provide context to the business concepts and tracks changes over time. Unlike the hub and link, the satellite is required to contain system version attributes. This enables the satellite to track data changes, and only allows data to be loaded if there is at least one change in the record (other than the system fields). Multiple satellite entries can describe one hub or one link, separated by date of change. In addition to the descriptive attributes, a satellite contain: the uniquely generated sequence id from the hub, or from the link to which it is attached, the timestamp when the data was loaded, and the source of the data.

2.7 Extract-Transform-Load (ETL)

The extract, transform, and load process refers to the set of activities responsible for bringing in the source data into the data warehouse. In the case of the DV-based DW, the first ETL process is responsible for loading the DV, and a subsequent ETL process responsible for loading one or more data marts from the DV. The ETL process is one of the back-room operations of the data warehouse, and is often a complex and labor-intensive undertaking.

CHAPTER 3
SOME COMMON DW ARCHITECTURES

An understanding, or at least an appreciation, of the different data warehouse architectures is required of the QA team, since the general testing strategy is impacted by the architecture selected for the implementation of the data warehouse. While an exhaustive treatment of the various DW architectures is beyond the scope of this work, the more common ones are highlighted here for reference purposes and for providing context in which the DW testing is done. For an elaborate treatment of the various DW architectures, the interested reader can refer to Golfarelli and Rizzi (2009a) which provides an excellent treatment on the subject.

In the scientific literature, two different classifications are commonly employed for addressing data warehouse architectures. The first classification relies on the number of layers used by the architecture. In the second classification, the emphasis is on the way the different layers are used to create enterprise-oriented or department-oriented views of the data warehouse.

3.1 Architectures Based on Number of Layers

3.1.1 Single-Layer Architecture

Single layer architecture is seldom used in practice. The goal of using this approach is to reduce the amount of data stored by removing data redundancies. It has only one physical layer and the data warehouse is considered virtual. A middle ware has to be employed to create a multi-dimensional view of the data, or an intermediate processing layer is necessary (Devlin and Cote, 1996). The inherent problem with this architecture is that there is no separation of operational data and analytical processes.

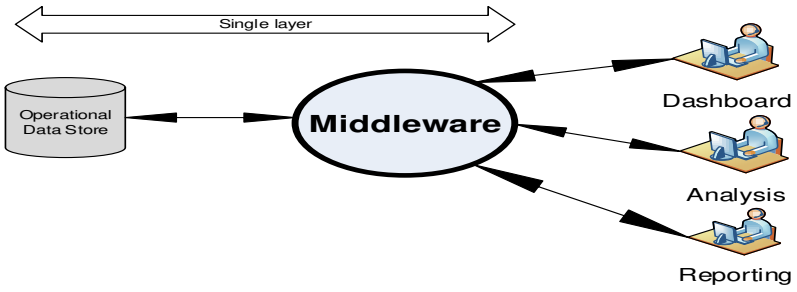


Figure 3-1 Single-layer architecture

3.1.2 Two-Layer Architecture

In the two-layer architecture, the requirement for the separation of operational and analytical processes is satisfied. The term "two-layer architecture" can be somewhat misleading because even though there are two distinct levels of separation (source and data warehouse), it actually consists of four stages of data flow as seen in figure 3-2 below.

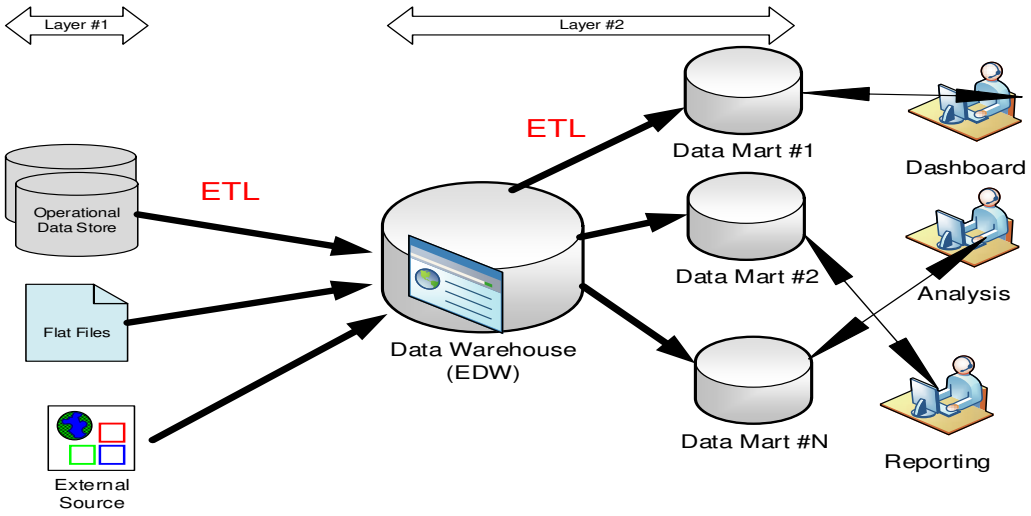


Figure 3-2 Two-layer architecture

3.1.3 Three-Layer Architecture

The three-layer architecture features a reconciled layer as the third layer. This new layer materializes operational data obtained after integrating and cleansing source data. The main advantage provided by the reconciled data layer is that it creates a common reference data model for the whole enterprise, while at the same time effectively separates the extraction and integration problems from those of data warehouse population (Golfarelli and Rizzi, 2009a).

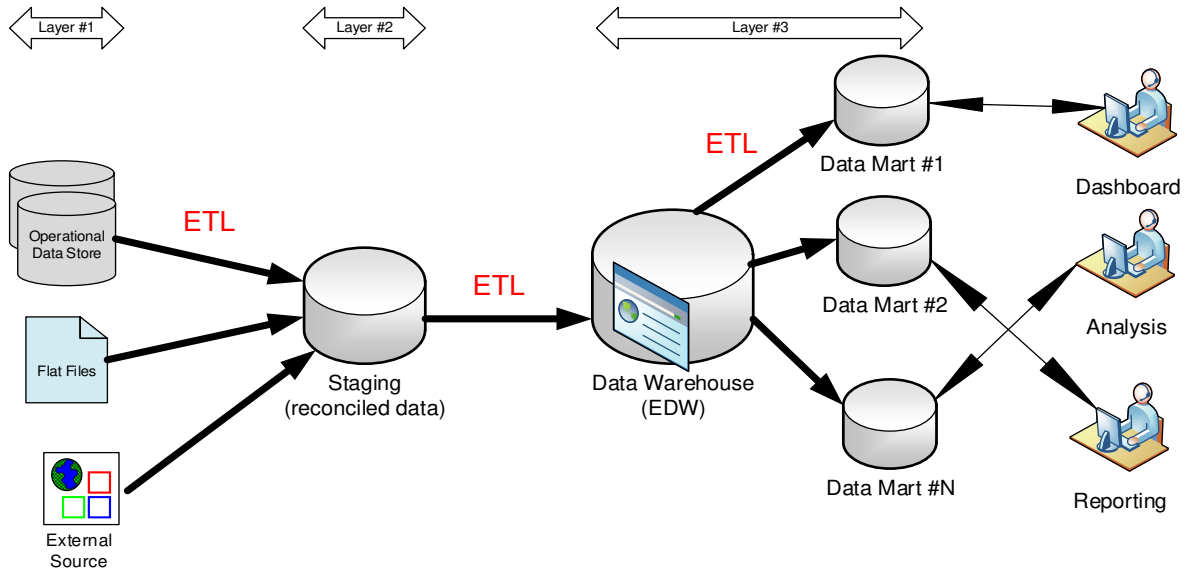


Figure 3-3 Three-layer architecture

3.2 Architectures Based on Enterprise-oriented View vs Department-oriented View

The layered approaches for classifying DW architectures (described previously), based on different combinations, are often used as the framework for providing an alternate classification for distinguishing five types of data warehouse architectures (Rizzi, 2008).

3.2.1 Independent Data Marts

The different data marts are designed and built in an independent non-integrated fashion. This approach seems attractive if organizational departments that make up the enterprise are loosely coupled, or if there is no strong support or sponsorship for the data warehouse project.

3.2.2 Bus Architecture

The Bus architecture is recommended by Ralph Kimball and is similar to the independent data mart architecture, but with one fundamental difference: the bus architecture employs a basic set of conformed dimensions which ensures the same meaning over all the facts they are associated with (Kimball and Ross, 2013).

3.2.3 Hub and Spoke Architecture

This architecture is one of the most used architectures for medium to large scale DW context. Scalability and extensibility are greatly emphasized for achieving enterprise-wide view of information. It uses a reconciled layer of normalized data which feeds a set of data marts. Although users mainly access the data marts, they may occasionally query the reconciled data as well.

3.2.4 Centralized Architecture

This architecture is recommended by Bill Inmon, (Inmon and Linstedt, 2014), and can be viewed as a special implementation of the hub and spoke architecture, whereby the reconciled layer and data marts are merged into a single physical repository.

3.2.5 Federated Architecture

The federated architecture is sometimes adopted for dynamic contexts where there are preexisting data warehouses and/or data marts and these are to be seamlessly integrated to provide a larger, single, organization-wide decision support environment. A typical example of when this approach is favored is in the case of a merger or an acquisition.

3.3 Where Does the Data Vault fit in the architecture?

A review of the various architectures for the data warehouse immediately reveals that the single-layer architecture is not applicable for a DV implementation, and is therefore not interesting in the context of this work. The item labelled “Data Warehouse, EDW” in both the two- and three-layer architectures represents the DV when the DV-based model is used. Whenever the DV is used in the context of the three-layer approach, it can benefit from the staging provided by the additional layer which takes care of the cleaning and integrating of data from the various, potentially heterogeneous, sources. The downside of this approach is not just the overhead of an additional layer; whenever loading errors are found the staging database has to be cleaned out and the loading process repeated. Also, since the data instances from the sources are reconciled, ability for tracking and auditing of source data items are further complicated. However, when the two-layer approach is used, the raw sources can be loaded directly into the data vault via an ETL process. This approach provides for greater investigation of the data vault for capturing data of less than ideal quality and allows progressive upgrades over time, when the correct values are known. All corrections are done via inserts, as no updates (in the sense of traditional SQL

UPDATE) are allowed for the DV-based model. The "reads all data - stores all data" bias of the DV allows for the source data to be as close as possible, in appearance, to the data in the DV. As a result, the DV model using the two-layer architecture is most interesting for this work. This approach is explored in details in chapter 8.

CHAPTER 4
THE QA ROLE AND COMPARING DB AND DW TESTING

4.1 Implications for the Quality Assurance (QA) Team

The role of the QA team is critical for diligent and thorough testing, and for ensuring that the data in the data warehouse is of the highest quality. The scope of the responsibility of the QA team is extremely wide, and exhaustive treatment lies outside the scope of this work. Though a full treatment is beyond the scope of this research, QA role is mentioned here to highlight how germane it is to general data warehouse testing.

The quality assurance analyst must possess a very good understanding of data modeling and source-to-target data mappings to be sufficiently equipped to develop an appropriate testing strategy. During the project's requirements analysis phase, the QA team must work to understand the technical implementation of the data warehouse. However, as outlined by the authors in (Rao, Ramesh, and Jamuna, 2009), because QA is a rigorous function that adds significant effort and expenditure to the software development cost, the QA process is oftentimes compromised. Because a data warehouse primarily handles data, a major portion of the test effort is spent on planning, designing, and executing data-oriented tests. In particular, the QA team should possess, according to (Yaddow, 2013):

- an understanding of the fundamental concepts of databases and data warehousing
- high skill levels with SQL queries and data profiling
- experience in the development of data warehouse test strategies, test plans, and test cases - what they are and how to develop them, specifically for data warehouses and decision support systems
- skills to create effective data warehouse test cases and scenarios based on business and user requirements for the data warehouse
- skills and interest in participating in reviews of the data models, data mapping documents, ETL design and ETL coding - as well as the ability to provide feedback to designers and developers
- knowledge of data transformation rules

4.2 Comparison of Database Testing and Data Warehouse Testing

People often confuse ETL testing with backend or database testing, but it is much more complex and different than that (Kamal and Nakul, 2010). The difference between a database and a data warehouse does not lie in just the volume of data, but also in the ETL process – a critical component of the data warehouse. As such, data warehouse testing should be aligned with the data model underlying a data warehouse. Database/generic software testing versus data warehouse testing depends on a number of factors as presented in (Golfarelli and Rizzi, 2009b; Mathen, 2010). A comparison of these factors is provided in table 4-1.

Database	Data Warehouse
Small to medium scale	Large scale. Voluminous data
Usually used to test data at the source instead of testing using GUI	Included several facets. Extraction, Transformation and Loading mechanisms being the major ones
Usually homogeneous data	Heterogeneous data involved
Normalized data	De-normalized data
Create, Read, Update and Delete operations (CRUD)	Usually Read-Only operations
Consistent data	Temporal data inconsistency
Limited use scenarios	Theoretically unlimited use scenarios (i.e. virtually any view of data)
Self-contained	Difficult to anticipate future requirements (i.e. never really come to an end)

Table 4-1 Comparative overview of database testing versus data warehouse testing

CHAPTER 5
GENERAL DW TESTING

Different stages of the data warehouse implementation - source data profiling, data warehouse design, ETL development, data loading and transformations, and so on - require the testing team's participation and expertise. Unlike some traditional testing, test execution should not start at the end of the data warehouse implementation. In short, test execution itself has multiple phases and should be staggered throughout the lifecycle of the data warehouse implementation (Mathen, 2010; Yaddow, 2013). Figure 5-1 below depicts how the testing is staggered for the data warehouse implementation. In addition, collaboration between IT and the business will help identify the reasonable issues versus the issues that cannot be resolved for the current release of the data warehouse. Prioritizing the test plan can help the testing team make these trade-offs (Goldman, 2007).

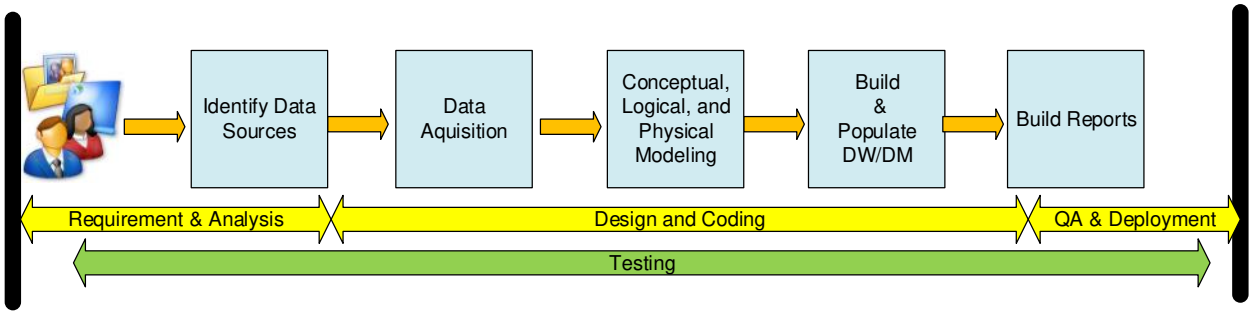


Figure 5-1 End-to-end data warehouse testing

Nonetheless, the data warehouse test plan need to address the six software quality factors of correctness, usability, efficiency, reliability, integrity, and flexibility (Golfarelli and Rizzi, 2009b). The types of testing tailored for the data warehouse environment can be summarized as follows:

5.1 Unit Testing

Unit testing is the process of validating each constituent part of the solution. This is the responsibility of the developer and must be done continuously during the development. In the context of a data warehouse, the most critical candidates for unit testing are ETL logic, business rules and calculations for the Online Analytical Processing (OLAP) layer.

5.2 System Integration Testing

System integration testing is aimed at confirming that the system acts as expected once the various parts of the solution are put together. With the raw DV load approach, the source data is loaded directly into the DV, so the integration of the various sources is pushed further downstream closer to the end users Data Marts.

5.3 Data Validation

Data validation is the process of testing the data within the data warehouse. Ad hoc queries are commonly used to retrieve data in a format that is similar to existing operational reports for comparison.

5.4 Security Testing

Security testing is geared towards checking how well the system protects the data and enforces the access rights.

5.5 Performance Testing

Performance testing is geared towards confirming that the system performs at or above agreed levels under normal workload conditions.

5.6 Regression Testing

Regression testing is the process of rechecking functionality to ensure that a recent change did not break features/functionalities previously known to work correctly.

5.7 Recovery Testing

Recovery testing is concerned with measuring how well the system is able to recover from a malfunction or crash.

5.8 User Acceptance Testing

User acceptance testing comprises verifying that the data provided to the end user is exactly what is expected and that the tools/features provided to the end users meet their expectations. If successful, the system should be acceptable to the user in terms of integrity of the ETL process, functionality, and correctness of the reporting.

CHAPTER 6
DW TESTING CHALLENGES AND GENERAL RECOMMENDATIONS

6.1 Challenges

Testing the data warehouse require overcoming a number of challenges. The most common hurdles are those of requirement for integrating heterogeneous sources of data, large volumes, and absence of standardized tools. Source systems data quality cannot be overemphasized. These issues can be further categorized as follows:

6.1.1 Challenges with Source Data

- Incorrect spelling
- Missing data
- Inconsistent data
- Invalid data
- Orphan records
- Invalid formats

6.1.2 Challenges With Integration

- Transformation errors due to faulty rules
- Invalid data
- Incomplete transfers

6.1.3 Challenges with BI

- Poor performance
- Scalability and upgrade issues
- Inaccurate reports

6.2 General Recommendations

Testing a business intelligence/data warehouse system is challenging. Standard testing methodology focuses on one thing at a time. However, integration and complexity are common themes of a BI/DW, which normally boasts voluminous data. As such, a different approach is required for testing the data warehouse. There are five top recommendations for building and executing a testing environment for the BI/DW and outlined by (Mundy, 2011):

6.2.1 Create a small static test database, derived from real data

The test database should be small so tests can run quickly. It should also be static so that the expected results are known in advance. Additionally, it is to be derived from real data because there's nothing like real data to give you a realistic combination of scenarios, both good and bad.

6.2.2 Test early and often

Start testing as soon as possible after writing a line of code or after connecting two boxes in your ETL tool's user interface. Developers are familiar with developing and running unit tests to ensure their code does what it's supposed to do. Unit testing assures that a developer's code works as designed. System testing ensures that the entire system works, end to end, according to specification. Start system testing early in the process, so all the kinks are worked out long before the pressure-cooker system testing phase begins.

6.2.3 Use testing tools and automate the test environment

Automate, automate, and automate. Testing early and often is practical only if the testing process is automated. But, automation requires tools. Some organizations will already have system quality assurance testing tools in place. If that is not the case, try searching the internet for "software quality assurance tools", which should bring back an overwhelming list of products and methodologies spanning a wide range of costs. Most, if not all, commercial test tools will allow you to perform functions such as enter tests, execute tests, log the results of test runs, and report on those results. For unit testing and data quality testing, define tests to run a query in the source and target data warehouse, then verify that row counts and amounts match up. Other important considerations for automation should include:

- running a script that sets up the test environment (or at least some portions of it)
- restoring a virtual machine environment with clean test data

- modifying the static test data with special rows to test unusual conditions
- running the ETL program

The traditional testing methodology usually changes one thing at a time (keeping all other variables unchanged), run a test and log the result. In the DW/BI world, expect to group together many tests to form a test group, because even with a tiny database, you don't want to execute your ETL code for each of the hundreds of unit tests that you should be running.

6.2.4 Enlist the business users to define system tests

The expertise of the business users should be harnessed to define good system tests. The familiarity of the business users with the process should not be overlooked in answering questions such as: How do we know the data is correct? How do we know that the query performance meets expectation? Including business users in the test specification process will ensure better testing results than if the DW/BI team work with tests made up solely on the basis on what they think is interesting. There is also the added benefit of credibility boost when business users engage in defining system tests.

6.2.5 The test environment must be as similar as possible to the production environment

While it is not always feasible to make the test environment identical to the production environment owing to budget constraints, every effort should be made to ensure the test environment is as close as possible to the production environment. Some elements of duplicity can be achieved without much additional costs and should be considered:

- as hard disk storage is relatively cheap, duplication of drive letters and storage location for database files are within reach and should be done
- use the same versions of operating systems, database, custom applications, and desktops
- security roles and privileges should be configured the same way in the test environment to allow for effective testing of access rights

CHAPTER 7

DATA QUALITY, MODELS, AND PATTERNS FOR RAW DV LOADS

In this chapter we look at data quality issues, data models and load patterns for DV loads. The load patterns presented in this section are for supporting raw data loads for DV 1.0 and DV 2.0, and differ in some respects from those that rely heavily on staging before DV loads.

The data models and load patterns presented in this chapter are used throughout the rest of this work, provide the framework for the experiments in chapter 8, and are based on a fictitious car rental company that has multiple agents (locations). The models have been simplified to remove much of the unnecessary details in an effort to foster greater focus on the concepts. For example, a number of the attributes have been omitted, and it is assumed that the vehicle is returned on time and at the same location it was rented. The vehicle pickup and return operations are not covered.

A subset of the relational model of the operational tables (simplified nonetheless) for the car rental company is provided below in figure 7-1 below.

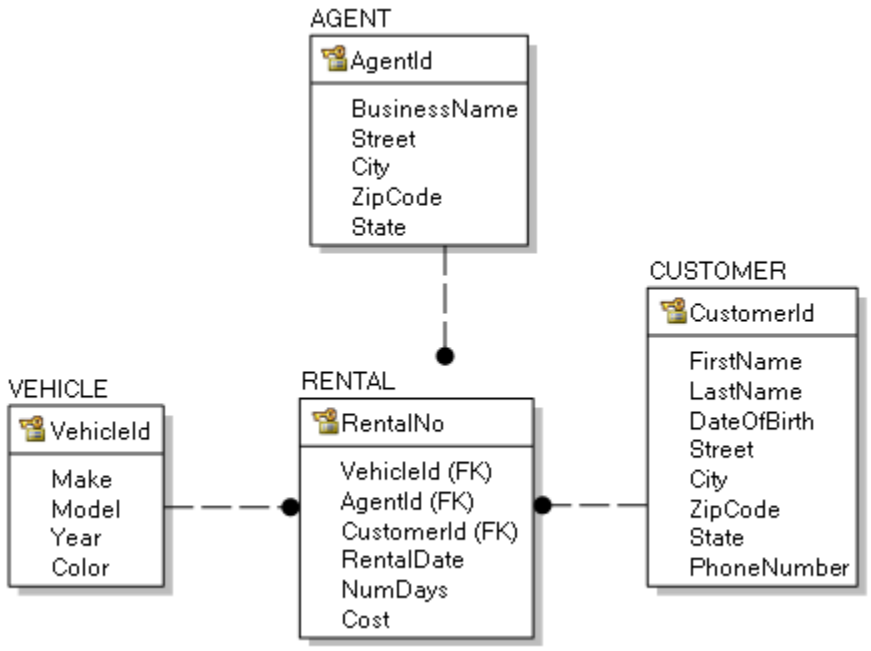


Figure 7-1 Relational Model of operational/source tables (simplified)

7.1 Model and Patterns for Raw DV Loads From Source Systems – DV 1.0

Figure 7-2 below shows the DV model (with the entities in the de facto coloring standard) for the car rental company. Most of the attributes are self-explanatory; however, the DV system-related metadata may not be quite obvious, and are as follows:

- *_Seq are the sequence generated surrogate keys. For example, Agent_Seq is the sequence surrogate key for the Agent business concept.
- *_Bk are the business keys mapped from the source. For example, Customer_Bk is the DV representation of the CustomerId attribute.
- LoadTimestamp is the date time stamp when record is loaded. It must be noted that for the Satellites, the Valid_From_Date attribute (i.e. the second attribute making up part of the primary key) is identical to the LoadTimestamp, so including LoadTimestamp is not absolutely necessary in these Satellites tables.
- LoadProcess is the ID of the ETL Load that has loaded the record
- RecordSource is the source of the record. At the minimum, it stores the source system name. However, it can be used to store more specific values such as the source table name.

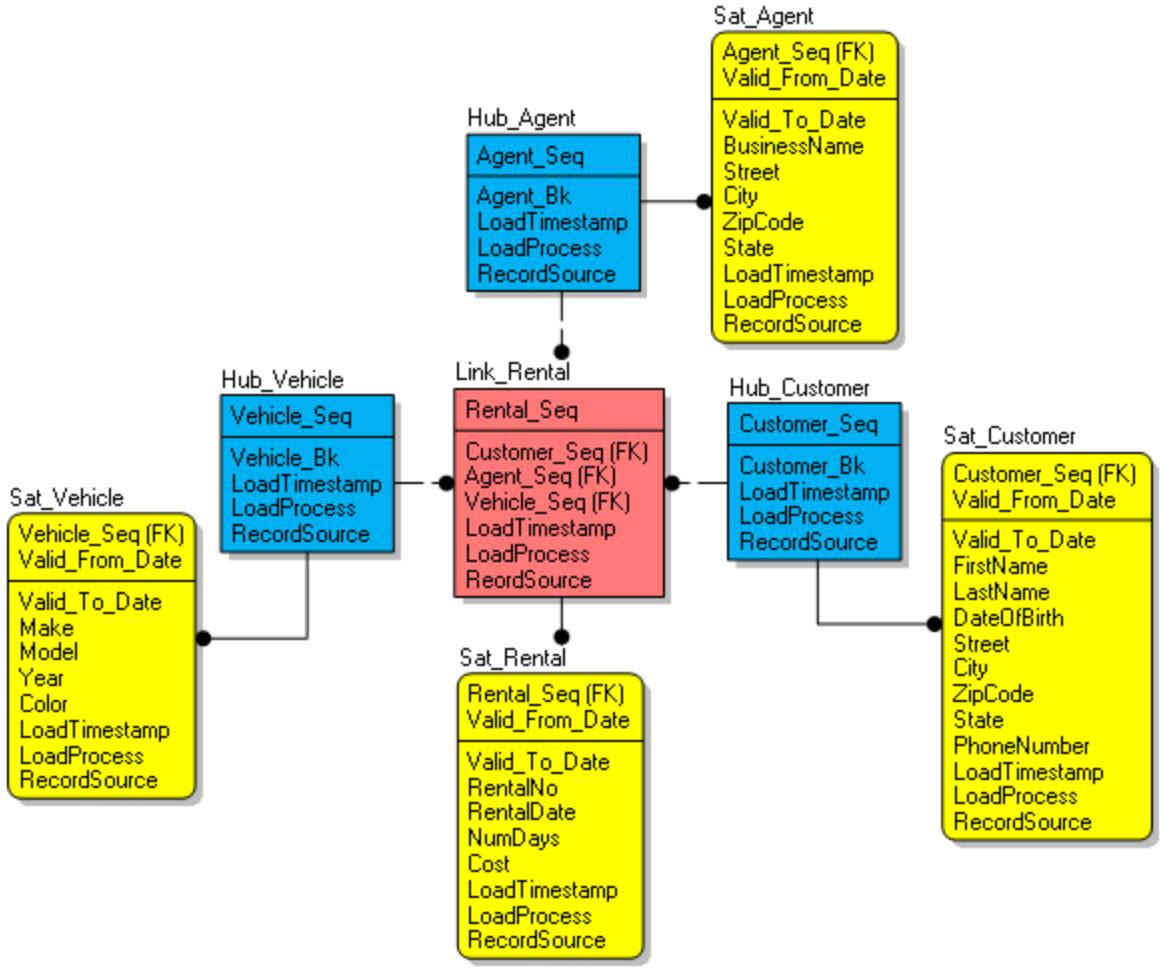


Figure 7-2 Data Vault (1.0) model for the Rental Company (simplified)

7.1.1 Load Patterns for DV 1.0

Hub Load

The concept behind Hub load is fairly straightforward as shown below:

(i) Generate list of distinct Business Keys

(ii) If a Business Key in the list exists in target Hub, then drop its row from the data flow

(iii) For each remaining row with distinct Business Key, generate Surrogate Key

(iv) Insert row into target Hub

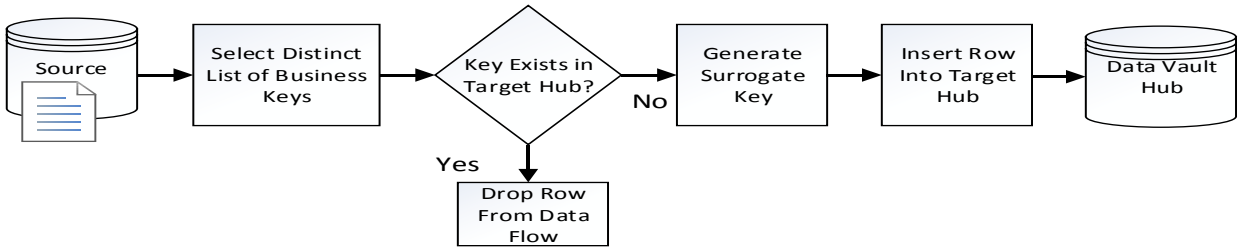


Figure 7-3 Data Vault (1.0) Hub Load Pattern

Link Load

The Link load bears some similarity to the Hub load. However, in DV 1.0 the Link load requires a lookup for the Surrogate Key for each Hub in the relationship modelled by the Link entity. Also note that the generation of a Surrogate Key for the Link is optional and depends on whether or not the Link will have associated Satellite to store attributes beyond the system attributes. The load pattern is shown below:

(i) Generate list of distinct Business Keys

(ii) Lookup each Hub's Surrogate Key

(iii) If the Surrogate Key exists in target Link, then drop row from data flow

(iv) For each remaining row (i.e. combination of Hub Surrogate Keys not found), generate Surrogate Key for Link record (if deemed necessary)

(v) Insert row into target Link

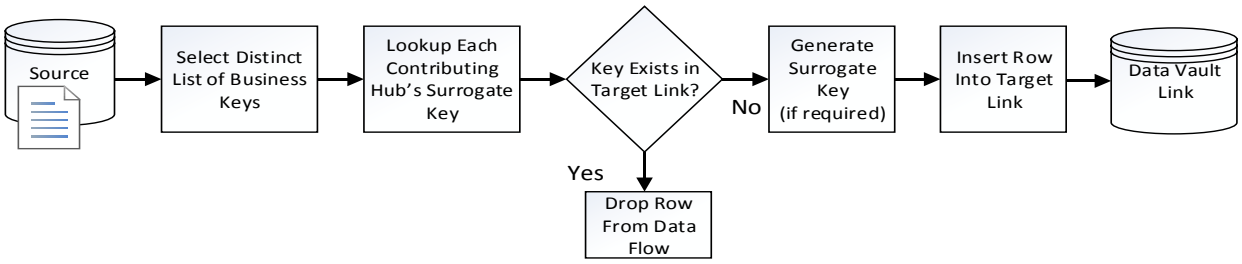


Figure 7-4 Data Vault (1.0) Link Load Pattern

Satellite Load

In the Data Vault, Satellites can be connected to either a Hub or a Link. Loading a Satellite connected to a Link differs from loading a Satellite connected to a Hub. In DV 1.0 the load for a

Satellite associated with a Hub requires a lookup for the Hub's Surrogate Key. However, the load for a Satellite associated with a Link requires lookup for each Hub contributing to the Link, as well as the extra lookup using contributing Hubs' combined Surrogate Keys to get the Surrogate Key of the Link. The Satellite end-dating, associated with Slowly Changing Dimension (SCD) Type 2 processing (Corr and Stagnitto, 2011), required for versioning in Satellites is included for completeness. The load pattern is shown below:

(i) Get distinct list of Satellite Records

(ii) For Hub-based Satellite, lookup Hub's Surrogate Key. For Link-based Satellite, lookup contributing Hubs' Surrogate Keys and use combination to lookup Link's Surrogate Key

(iii) Find latest Satellite record

(iv) If no Satellite record found, simply insert row in target Satellite.

(v) If 'latest' Satellite record found and all fields/columns match, drop row from data flow. Else, if "latest" record found and does not have a match on all field/columns, insert row in target Satellite and perform end-date processing for record previously identified as "latest" record, to indicate it is no longer the most current record.

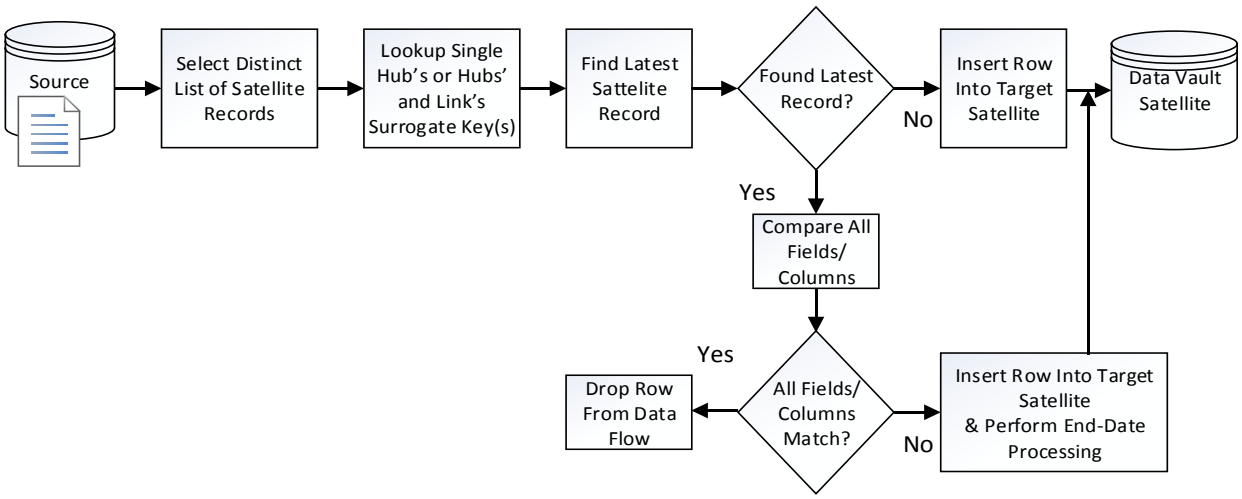


Figure 7-5 Data Vault (1.0) Satellite Load Pattern

Figure 7.6 below (taken from actual SSIS ETL Load Job in chapter 8) reveals that DV 1.0 requires at most three waves to load:

(i) Hubs - generate hub surrogate keys

(ii) Links - requires Hub surrogate keys as foreign keys, and to build a Link surrogate key (when required)

Hub Satellites - requires Hub surrogate key to combine with Valid_From_Date to form primary key

(iii) Link Satellite - requires the Link surrogate key to combine with Valid_From_Date to form primary key

Refer to Appendix D for full ETL Control Flows and Data Flows.

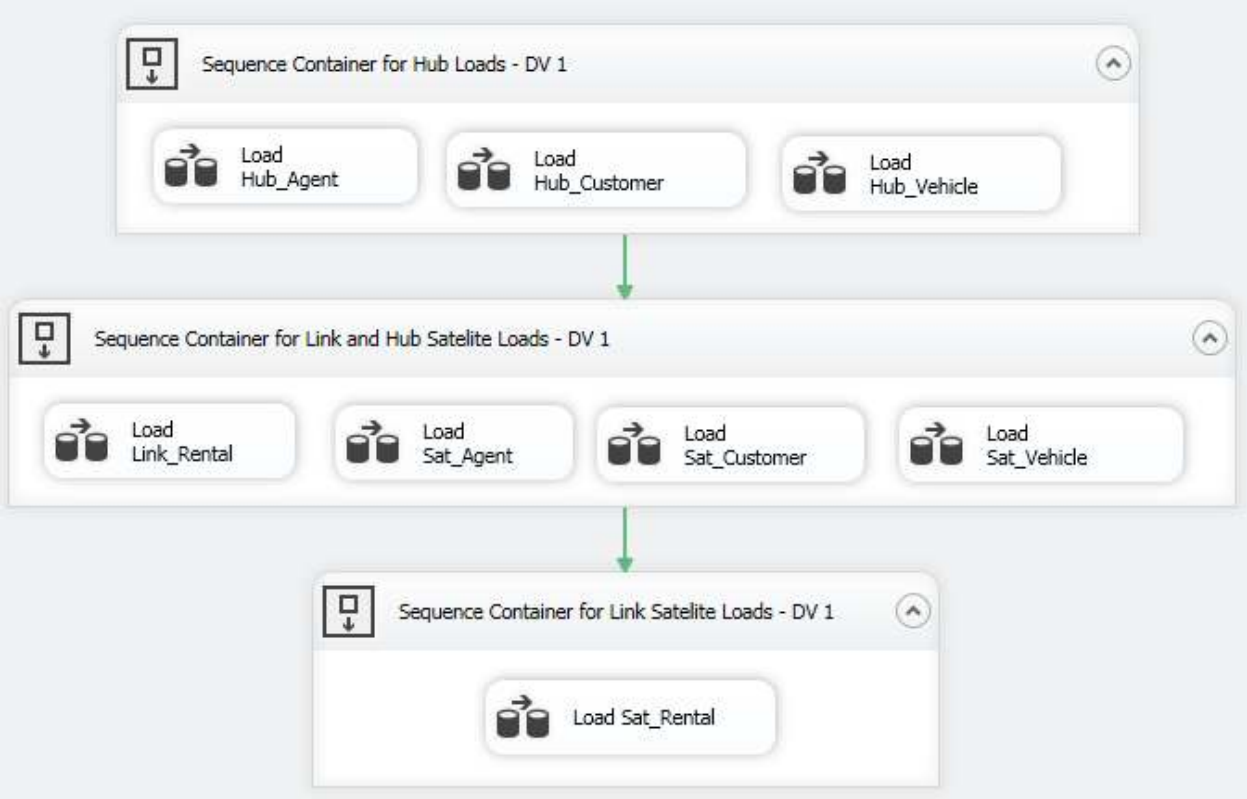


Figure 7-6 SSIS ETL Control Flow for DV 1.0 load showing three waves enforced by Sequence Containers

7.2 Model and Patterns for Raw DV Loads From Source Systems – DV 2.0

DV 2.0 is an update to the original concept of the data vault, DV 1.0, created by Dan Linstedt. Whereas DV 1.0 is highly focused on the Data Vault Modelling components (Hubs, Links, and Satellites), DV 2.0 features a number of development areas, just to name a few, such as:

- Agile development methodology
- Adaptation to support NoSQL databases

- Data Vault methodology
- Data Vault and data warehouse architecture

However, in addition to the larger areas of interest, there is a concrete concept which is worth exploring – the use of hash primary key (PK) in hubs and links instead of the traditional surrogate keys using sequence numbers. The change is motivated by a number of factors (Inmon and Linstedt, 2015):

- The efforts to connect heterogeneous data environments such as Hadoop
- To remove the dependency on loading DV 2.0 structures, especially when dealing with high ingestion rates (velocity) or big data (variety and volume)
  With the removal of the dependency during loading, greater parallelism can be achieved

7.2.1 Using Hash Keys in the DV

A hash function maps data of arbitrary size to data of a fixed size. The resulting hash value is in binary, however, it is commonly represented in hexadecimal to make it more human readable. Hashing is appealing because it is very difficult to reverse a hash to obtain the original value (i.e. hashing is a one way process). This means that sensitive data items like a password can be stored in hash value representation (for privacy concerns) and comparisons done by first hashing the newly input password and comparing the hashes. Another interesting feature of hashing is that two same inputs will always produce the same hash keys irrespective of the platform, and two different inputs will theoretically always produce different hash keys. In reality we could have repeats (collisions) for different inputs, however, with 128 bit (16 byte) hash keys the chance of repeats is as small as $\left(\frac{1}{2}\right)^{128}$. Whenever there is an extremely high requirement for uniqueness, as in the case of massive satellites, the SHA256 or MD5 hash function can be used, which creates 256 bit (32 byte) hash, and has an even less likely chance of repeats of $\left(\frac{1}{2}\right)^{256}$. Also, the same principle behind the comparison of the one-way hash of passwords can be used to detect changes on Slowly Changing Dimension (SCD) Type dimensions, using the concept of versioning.

Perhaps one of the greatest benefits of incorporating hashing in the data warehouse is that we can replace each traditional sequence number surrogate key with a hash key. The hash key is calculated using the business key as input to the hash function. Popular ETL tools such as Microsoft SSIS and SAS DI Studio store the business key (which can be single attribute or composite) within each satellite and link load job. Traditionally, the business key is used in a lookup operation on a hub or link table. It is interesting to note, as mentioned earlier, that a given input to a hash function will always produce the same output. As a result, there is no longer a need to do any lookup – the key (hash) to be found from the lookup can instead be calculated by running the business key through the hash function locally in the job. The benefit of this is the removal of load order constraints and opens up possibilities for more parallel loads.

For a given data vault, the actual DV model changes only slightly from DV 1.0 to DV 2.0, by replacing the surrogate sequence Id with a hash key. However, for completeness (especially for highlighting the naming convention change for keys from *_Seq to *_Hsk), the equivalent DV 2.0 model for the car rental company is shown below in fig 7.7. It is also worth noting that the physical schema will reflect a change from a surrogate sequence key represented as an Integer or BigInteger (depending on the target system) to a hash key represented as Char (32) for both SHA256 and MD5 hashes.

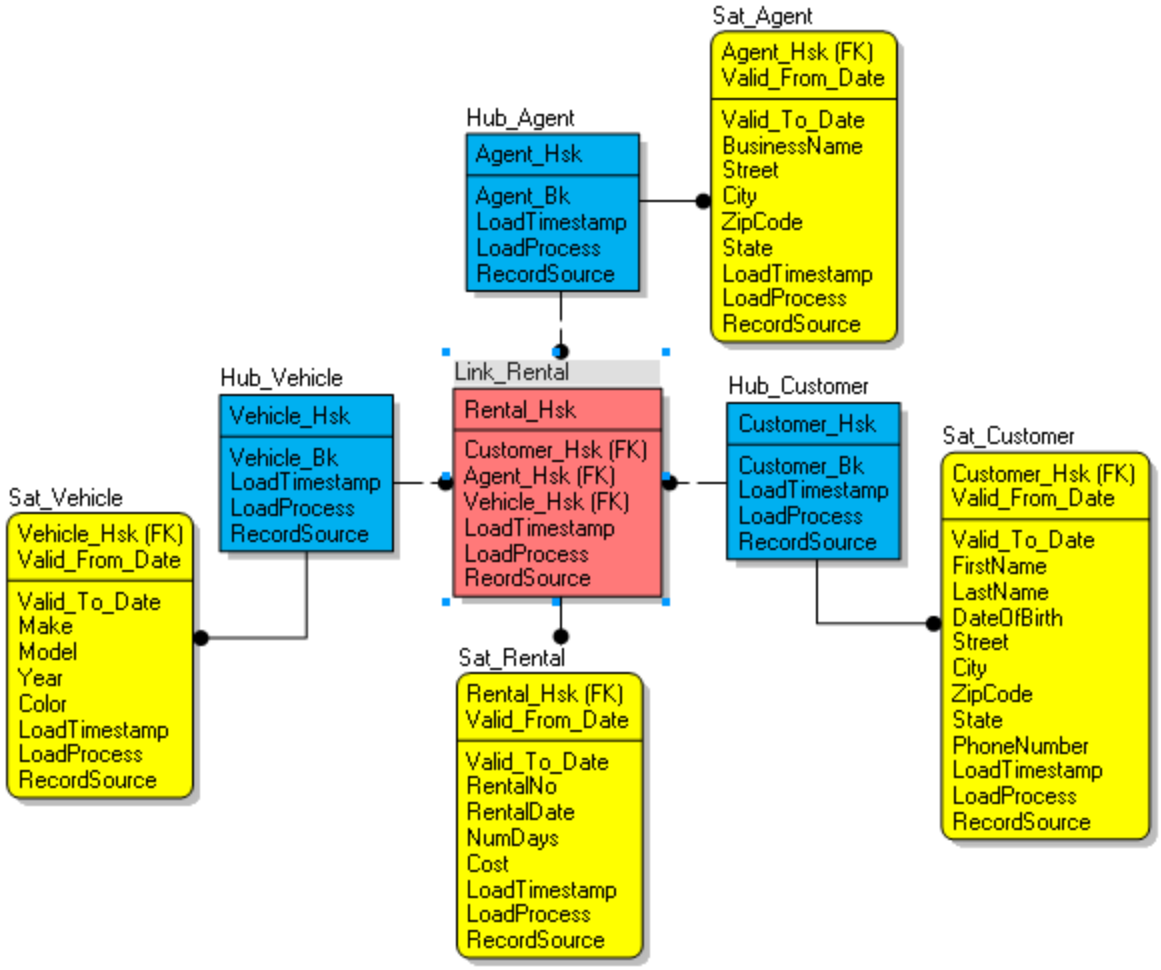


Figure 7-7 Data Vault (2.0) model for the Rental Company (simplified)

7.2.2 Load Patterns for DV 2.0

The load patterns for DV 2.0 are further simplified by the replacement of the surrogate sequence keys by hash keys, allowing the workflows to contain less dependencies and transformations.

Hub Load

The concept behind Hub load remains straightforward as shown below:

(i) Generate list of distinct Business Keys

(ii) If a Business Key in the list exists in target Hub, then drop its row from the data flow

(iii) For each remaining row with distinct Business Key, generate Hash Key using Business Key as Input

(iv) Insert row into target Hub

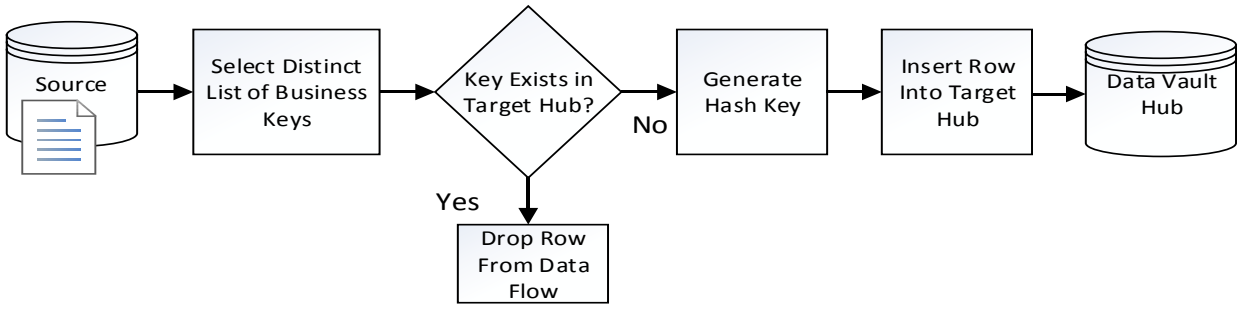


Figure 7-8 Data Vault (2.0) Hub Load Pattern

Link Load

The benefit of using hash keys was not quite obvious in the Hub load. However, for the Link load, the replacement of each lookup with the hash key generation will serve to provide savings in the form of reduced input/output cycles. The load pattern is shown below:

(i) Generate list of distinct Business Keys

(ii) Generate Hash Key for each Business Key

(iii) If the Hub Hash Keys exist in target Link, then drop row from data flow

(iv) For each remaining row (i.e. combination of Hub Hash Keys not found), generate Hash Key for Link record (if deemed necessary) by concatenating Hub Hash keys and running result through hash function

(v) Insert row into target Link

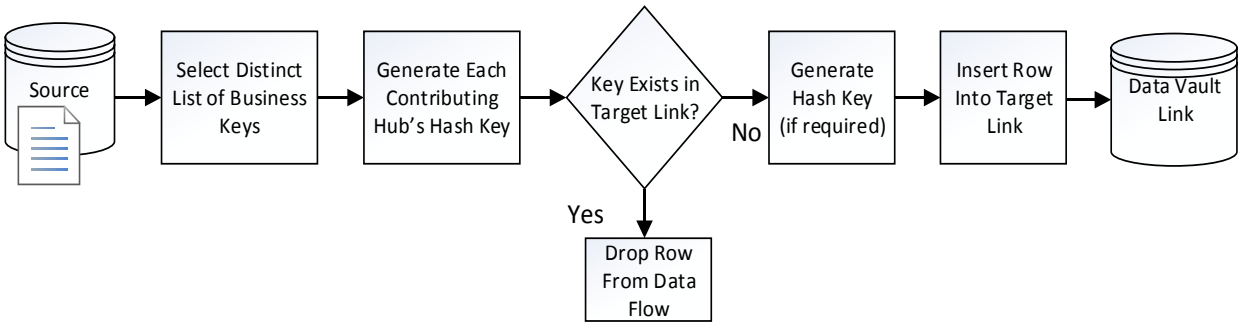


Figure 7-9 Data Vault (2.0) Link Load Pattern

Satellite Load

The benefits of using Hash keys are also extended to Satellite loads for the same reasons provided above. The load pattern is shown below:

(i) Get distinct list of Satellite Records

(ii) For Hub-based Satellite, generate Hash Key from Hub's Business Key. For Link-based Satellite, generate Hash Keys for all contributing Hubs' Business Keys, then concatenate keys and run though hash function to regenerate Link's Hash Key

(iii) Find latest Satellite record

(iv) If no Satellite record found, simply insert row in target Satellite.

(v) If 'latest' Satellite record found and all fields/columns match, drop row from data flow. Else, if "latest" record found and does not have a match on all field/columns, insert row in target Satellite and perform end-date processing for record previously identified as "latest" record, to indicate it is no longer the most current record.

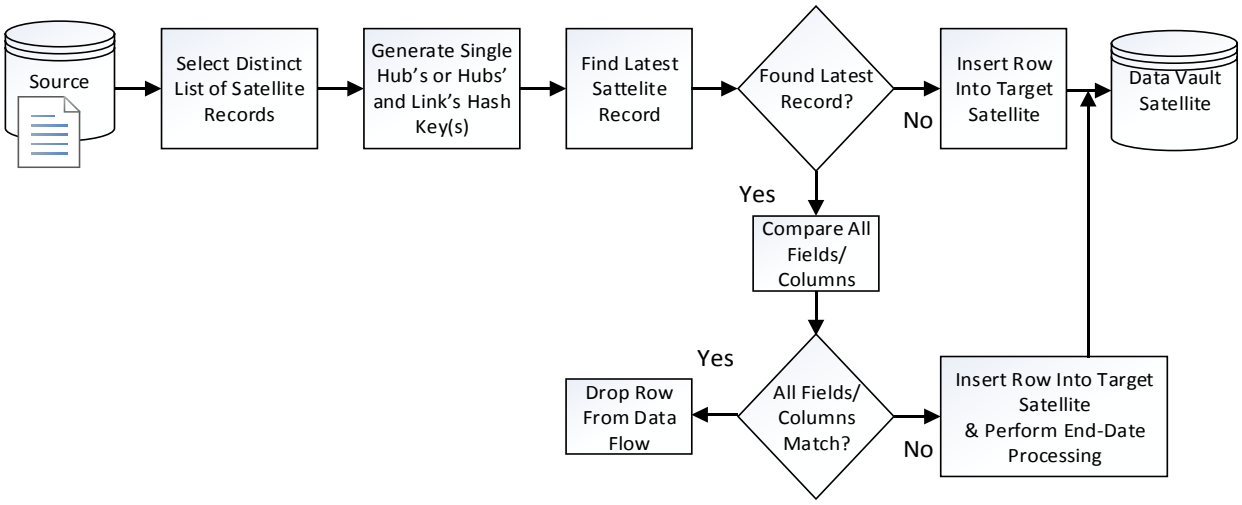


Figure 7-10 Data Vault (2.0) Satellite Load Pattern

Figure 7-11 shown below (taken from actual SSIS ETL Load Job in chapter 8) shows DV 2.0, with the introduction of hash keys and the resulting benefit of more parallel loads, using only two waves to load:

(i) Hubs - generate hash key from business key

(ii) Links - generate hash key from concatenation of contributing Hubs' hash keys
Hub Satellites - generate hash key from business key and combine with Valid_From_Date to form primary key
Link Satellite (generate hash key from concatenation of contributing Hubs hash keys and combine with Valid_From_Date to form primary key)

Refer to Appendix D for full ETL Control Flows and Data Flows.

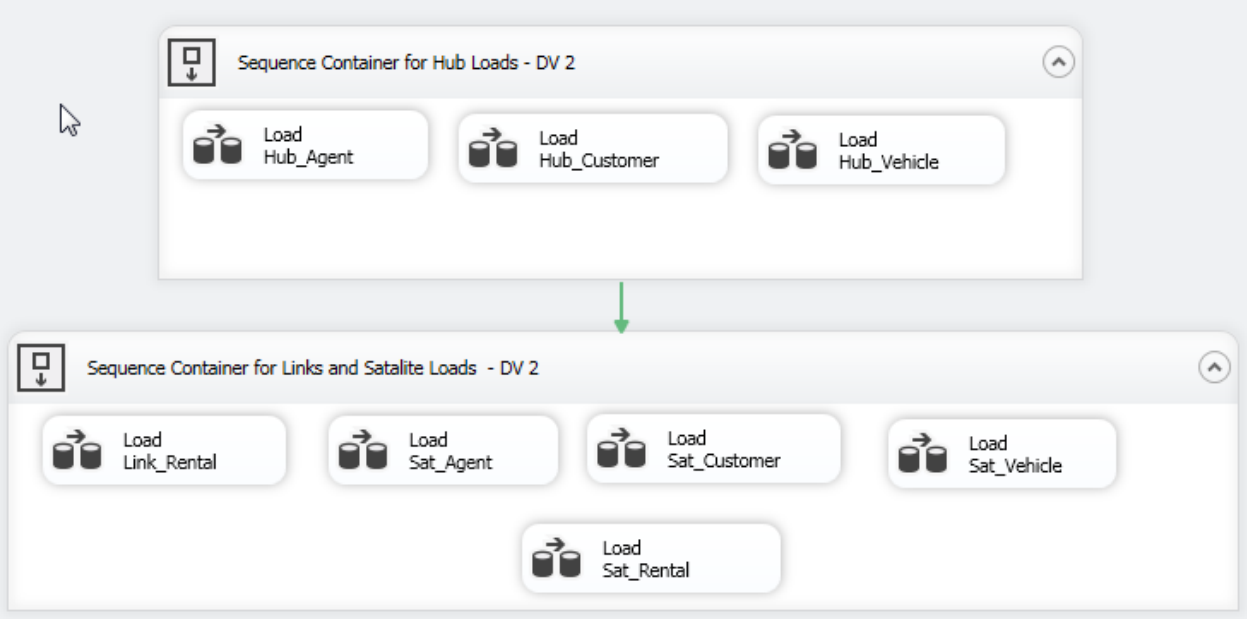


Figure 7-11 SSIS ETL Control Flow for DV 2.0 load showing two waves enforced by Sequence Containers

7.3 A Sample Data Mart loaded from the DV

Figure 7-12 below shows the Star Schema which could emerge to satisfy the requirements for a Data Mart to be developed for a particular department or business unit (and used in the experiments in chapter 8). Since each Data Mart represents only a subset of the enterprise data in the Data Vault, the selected columns and granularity (level of aggregation) are specific to the needs of the users of a particular Data Mart. For example, this Data Mart does not contain street addresses and RentalNo is filtered out, which are indicative that these columns are not deemed necessary for the reporting and analyses activities of the targeted end users. It is also worth pointing out that similarly to the DV, there are metadata columns for tracking record source and versioning in dimension tables via type 2 SCD. However, the system/metadata columns were intentionally left out of the diagram in order to present a simplified Star Schema which focuses on the columns/fields of interest for reporting and analysis activities. For audit metadata details see Appendix A for the SQL DDL script used to create the Data Mart.

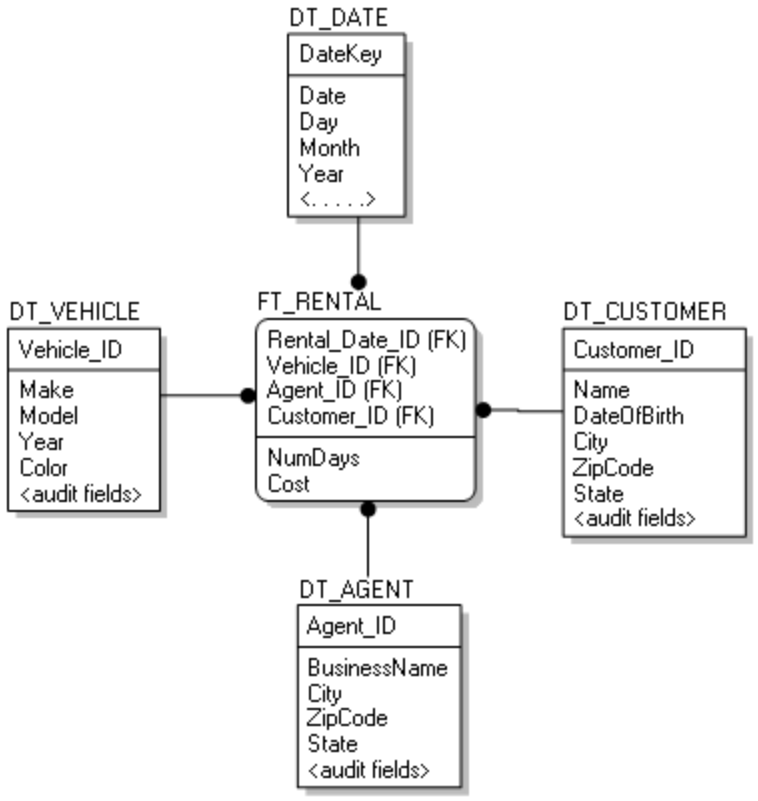


Figure 7-12 Star Schema for Data Mart (audit metadata column names omitted)

CHAPTER 8
RAW DV LOADS WITH PERMISSIBLE ERRORS

This chapter is a culmination of the models, load patterns, and concepts presented in earlier chapters. Through experimentation, it explores the novel concept and primary focus of this research:

> Raw DV loads from source systems, keeping transformations to a minimum in the ETL process which loads the DV from the source. Certain load errors, classified as permissible errors and enforced by business rules, are kept in the Data Vault until correct values are supplied. Major transformation activities are pushed further downstream to the next ETL process which loads and refreshes the Data Mart from the Data Vault.

8.1 The Problem

The traditional approach for dealing with errors during the data warehouse load normally results in one of the following actions:

(a) Fail the component/activity

(b) Ignore the error

(c) Redirect the error rows/records

Failing the component/activity aborts the load process and has the potential to cause delay to the DW project due to constant tear down/rollback and restarts when errors are detected. Simply ignoring error without follow up can be problematic - there is no feedback to the source provider about the errors so that corrective actions can be taken. Moreover, depending on the nature of the error, it may severely impact the quality of data fed downstream to data marts for reporting and analyses. Redirecting the error rows/records to an error report file is prudent; however, this action by itself prevents the DV from carrying out its role as the "capture-all store-all" enterprise data warehouse. The data in the DV must be truly reflective of the source data (Linstedt and Graziano, 2011). Aggressive cleaning of the source will go against a basic requirement of the DV: a comprehensive DW staging area able to store not only current data, but also past data without any alterations or loss (Jovanovic, et al 2012). Operational data errors must be kept on file in the DV and correction applied as new records instead of overwriting the errors - a crucial

requirement for industries such as banking and healthcare. Finally, the traditional loading approach also limits the extent to which an agile methodology can truly be applied to the DW project (Hughes, 2015), since Data Marts load and refresh testing are delayed until the staging and source loading issues have been resolved.

A main goal of this research was to develop and demonstrate a flexible approach, with repeatable load patterns for raw DV loads (both DV 1.0 and DV 2.0) from source systems, to address the issues stated above.

8.2 Solution Design

8.2.1 Hardware and Software Specifications

All tests and experiments were carried out on the same computer to ensure there was no bias for any given approach or test run. The hardware and software specification are summarized in table 8-1 below:

Operating System	Windows 7 Professional 64 bit
Processor	Intel Core i7 4702MQ CPU @ 2.20 GHz
Memory	8.0 GB
DBMS	SQL Server 2012 for DW and MySQL Community Server 5.6.24 for test data generation
ETL Tool	Microsoft SQL Server Integration Services (SSIS), SQL Server Data Tool for Visual Studio 2012

Table 8-1 Hardware and Software specification

8.2.2 Test Data and Test Data Generation

The Date dimension used in the Data Mart is based on the Kimball date dimension, with minor modifications. The date dimension was pre-populated with 10 years of data (from 2010-01-01 to 2020-12-31) and each record represents a calendar date within that period (including February 29 for leap years). Additional records representing dates in the past or present can be loaded at will to extend the date range.

All remaining test data used in this research were generated by the generatedata tool ("generatedata", 2015). The tool features a web form where the user specifies each column name and its corresponding data types. It supports generating data in various formats such as Excel, HTML, CSV or SQL format. However, the CSV and SQL format were sufficient for this research. The country-specific-data feature also allows for specific address, zip/postal code, and last names and first names (for example western-based names) formats. Generating datasets with error records deliberately and randomly injected was also fairly straightforward using this tool.

It is worth pointing out that even though elements of the test data such as first names, last names, cities, and zip codes seem real, the dataset consisted of fake data and randomly generated. This was evident in the fact that Customers were sometimes assigned addresses with a number range in the street number, and Rental Agencies were sometimes assigned addresses with P.O Box numbers. However, the street names were treated as mere labels and had no negative impact on the research. Figure 8.1 below shows the form used to generate rental transactions:

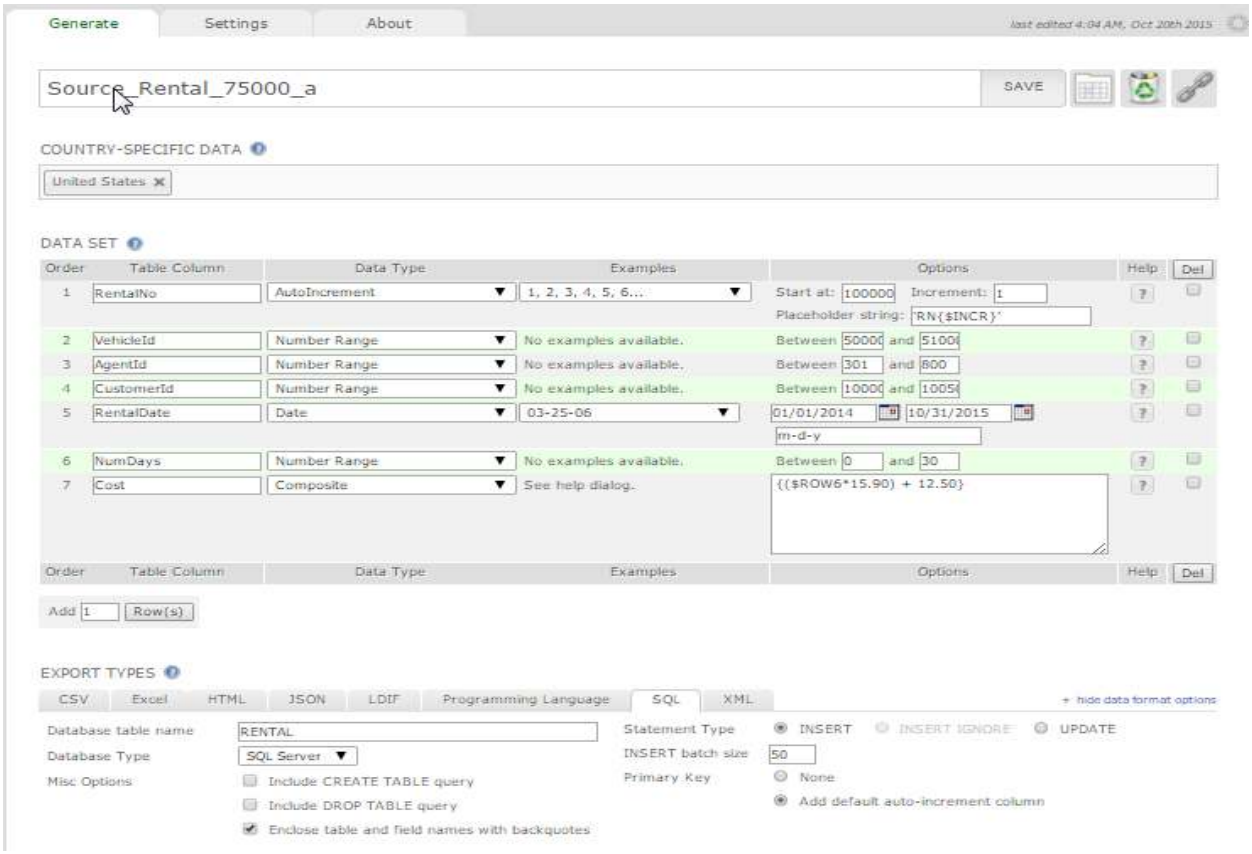


Figure 8-1 Web form for data generation of Rental transactions using generatedata tool

8.2.3 Data Scoping, Validation Rules, and Permissible Load Errors

Source data quality is dependent to a large extent on the governance of the schema and integrity constraints. For example, flat files, which have no schema, are more prone to erroneous data than their relational database counterparts (Rahm and Do, 2000). The concept of permissible load errors applies only to data errors caused from operational errors and not submission or system errors. Permissible load errors are essential for historical tracking and extended audit support. In contrast, a record with a fatal error such as missing business key cannot be permitted in the data warehouse because the business key is required for data integration. Since each DW implementation will have its own unique requirements, the set of permissible load errors will have to be determined on a case-by-case basis during the data scoping phase. The following business rules for permissible load errors were applied for this research:

Customer – DateOfBirth field checked: Is customer at least 18 yrs. old at time or rental?
Vehicle – vehicle's Year field checked: Is vehicle Year less than 3 years or more than 2 years the year of rental transaction?
Rental – RentalDays field checked: Is rental for a period of 0 to 30 days?
Data errors not listed above are treated as non-permissible errors and are rejected by both approaches.

8.3 Experiments

The generatedata tool was used to generate the following test records:
Agent: 500, Customer: 5,000, Vehicle: 10,000
Rental: 1,000, 10,000, 50,000, 75,000 and 100,000 (for the five iterations of load strategy and DV model combinations), with 1% of records containing permissible load errors.
Only one test was run at any given point in time, and the average time for three runs recorded.

A total of eight SSIS packages were used to carry out the various ETL operations:

(a) DV1_EDW_Trad_Load.dtsx – Loads DV 1.0 from source rejecting all load errors

(b) DV1_DM_Trad_Load.dtsx – Loads DM from DV 1.0 rejecting all load errors

(c) DV2_EDW_Trad_Load.dtsx – Loads DV 2.0 from source rejecting all load errors

(d) DV2_DM_Trad_Load.dtsx – Loads DM from DV 2.0 rejecting all load errors

(e) DV1_EDW_Load.dtsx – Loads DV 1.0 from source allowing permissible load errors

(f) DV1_DM_Load.dtsx – Loads/Refreshes DM from DV 1.0 skipping over records with permissible load errors and dependencies until corrections are supplied to the DV

(g) DV2_EDW_Load.dtsx – Loads DV 2.0 from source allowing permissible load errors

(h) DV2_DM_Load.dtsx – Loads/Refreshes DM from DV 2.0 skipping over records with permissible load errors and dependencies until corrections are supplied to the DV

Permissible load errors are kept in the DV for historical purposes, but cannot be transmitted to the DM as doing so would skew reporting/analyses results. Any dependent records, which may or may not themselves contain permissible errors, must also not be loaded or refreshed in the DM by the ETL job, as this could lead to sparse cube condition for any Relational Online Analytical Processing (ROLAP) implementation. In addition, each record in a fact table needs to have matching dimension records to maintain referential integrity in the DM. To enforce this requirement, foreign key constraints were implemented on the Rental fact table.

The ETL logic for preventing permissible load errors and dependent records from being sent to the DM is fairly straightforward: it involves applying the negation of each business rule (initially) used for allowing the permissible errors in the DV during the source-to-DV load phase.

After corrections are made, target records and all dependencies are sent to the DM in the next ETL load job. As an illustration of this principle, note below in figures 8-2 and 8-3 that customer Casey Mack's DateOfBirth was incorrectly captured as 1999-09-29 (instead of 1969-09-29), which would suggest that she was only 14 - 15 yrs. old at the time or her vehicle rental activities. Even though there are no errors in these rental transaction records, these records must not be loaded in the Rental fact table in the DM until after the correct DateOfBirth is supplied to the DV, and the Customer dimension table loaded or refreshed.

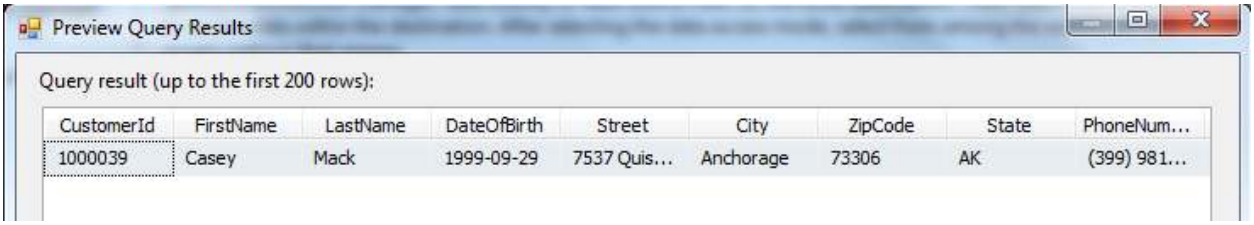


Figure 8-2 Query Preview Error in SSIS ETL package

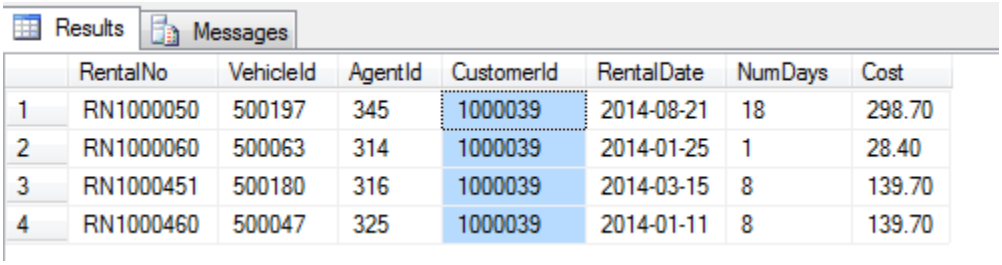


Figure 8-3 Query of transaction source

8.3.1 Traditional Approach – No Load Errors permitted in Data Vault

Records detected with errors during DW load were reported in an Excel spreadsheet for error correction and resubmission. Since the load job failed causing the process to rollback, the entire dataset with the corrections included was resubmitted for loading into the DV. The corrected dataset was subsequently used to feed the data mart downstream in the DM initial load. The load times with increasing number of Rental transactions are shown below in tables 8-2 and 8-3:

	DV Load		DM Load	**Total**
Rental Rows	Initial	Resubmit	Initial	
1,000	16.273	13.213	5.210	**34.696**
10,000	28.595	27.223	5.803	**61.621**
50,000	207.672	207.575	8.572	**423.819**
75,000	424.667	420.812	11.372	**856.851**
100,000	743.781	743.392	14.932	**1502.105**

Table 8-2 Load times for DV 1.0-based DW using “No Load Errors Allowed” strategy

	DV Load		DM Load	**Total**
Rental Rows	Initial	Resubmit	Initial	
1,000	7.934	5.913	6.942	**20.789**
10,000	8.659	6.880	7.269	**22.808**
50,000	103.803	103.522	10.405	**217.73**
75,000	232.831	231.974	11.138	**475.943**
100,000	424.986	421.187	14.258	**860.431**

Table 8-3 Load times for DV 2.0-based DW using "No Load Errors Allowed" strategy

8.3.2 Alternative Approach – Keeping permissible Load Errors in Data Vault

Records detected with errors during DW load were reported in an Excel spreadsheet for error correction and resubmission, as in previous approach. However, errors predetermined as permissible/allowable errors during DV load were accepted and held for future correction – a stark contrast to the previous approach. Additionally, the resubmitted dataset included only the corrected records. The load times (in seconds) with increasing number of Rental rows are shown below in tables 8-4 and 8-5:

	DV Load		DM Load		**Total**
Rental Rows	Initial	Retry	Initial	Refresh	
1,000	15.849	1.389	3.822	1.700	**22.760**
10,000	26.941	1.607	5.382	2.487	**36.417**
50,000	200.555	4.509	8.003	6.133	**219.200**
75,,000	435.727	8.860	10.389	8.502	**463.478**
100,000	733.686	12.792	15.168	10.874	**772.52**

Table 8-4 Load times for DV 1.0-based DW using "Permissible Load Errors" strategy

	DV Load		DM Load		**Total**
Rental Rows	Initial	Retry	Initial	Refresh	
1,000	6.017	1.154	4.804	1.685	**12.76**
10,000	7.269	1.981	4.961	2.528	**16.739**
50,000	103.787	3.962	9.251	6.240	**123.24**
75,000	236.151	5.725	10.890	8.455	**261.221**
100,000	420.625	9.906	15.959	10.904	**457.394**

Table 8-5 Load times for DV 2.0-based DW using "Permissible Load Errors "strategy

8.3.3 Analysis of Results

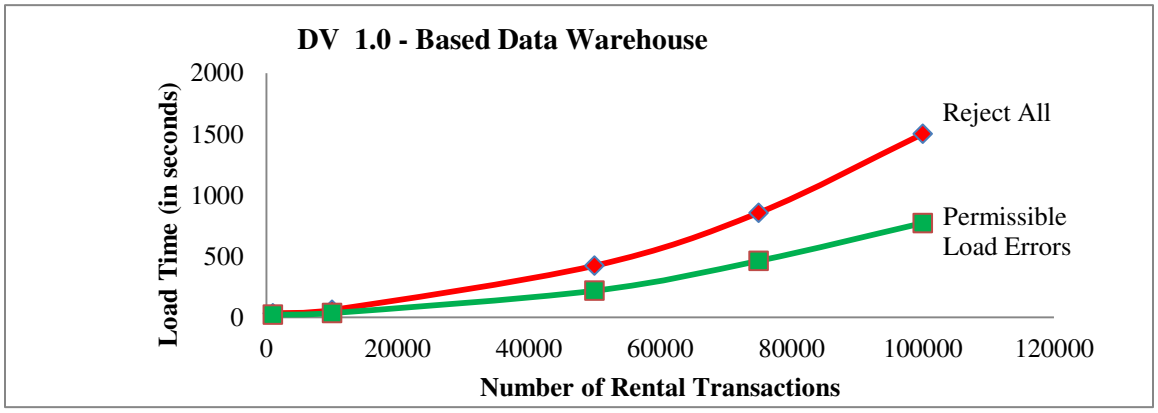


Figure 8-4 DV 1.0 – based DW Load times for "Reject All Load Errors" vs "Permissible Load Errors" strategies

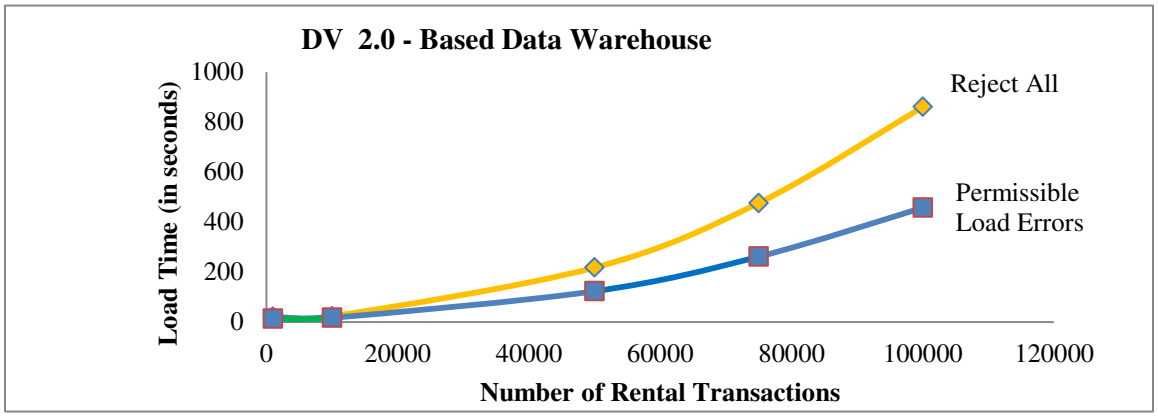


Figure 8-5 DV 2.0 – based DW Load times for "Reject All Load Errors" vs "Permissible Load Errors" strategies

Examination of tables 8-2, 8-3, 8-4, and 8-5 reveals that for both strategies, the load times for the data vault are much smaller for DV 2.0 than DV 1.0 for a given rental data set. This can be attributed to fact that DV 2.0 uses hash keys (generated from the business keys), thereby obviating the need for expensive lookups when loading satellites and links, and enabling more parallelism in DV loads. As seen in tables 8-2 and 8-3, the "No Load Errors Permitted" strategy generally requires more time for the initial DV load attempt than the resubmit attempt. This is due to the additional expense of writing out error report to Excel file. The "Permissible Load Errors" strategy is noticeable more expensive for loading the data mart from DV 2.0 than from DV 1.0. This can be explained by the joins required by the ETL logic, which uses a 32-byte hash keys in the case of DV 2.0 which is slower than using the 8-byte integer surrogate sequence used in DV 1.0. The total load times in the tables, further supported by the graphs in figures 8.4 and 8.5, reveal that the "Permissible Load Errors" strategy consistently provides significantly better load times than the traditional strategy for both DV 1.0 and DV 2.0. Moreover, larger datasets provide even greater gains in load times.

CHAPTER 9
CONCLUSION

One of the greatest risks to the successful implementation of a business intelligence system is putting the data warehouse into service without it being effectively tested by qualified QA and ETL testing teams. Everything in business intelligence revolves around data, so the simplistic approach of concentrating the testing effort at the implementation phase will not suffice for data warehouse projects. The complexity of data warehousing projects and the high cost of correcting errors later as opposed to sooner in the development life cycle, demand active participation of the testing team from the earliest phases of the project – the requirement gathering and design phase.

Using a DV-based data warehouse does not obviate the need to incorporate best practices testing for the non-DV portions close to the end user (such as the data marts), and general testing strategy as presented does apply. However, we have shown that by adopting a Data Vault-based Enterprise Data Warehouse we can simplify and enhance various aspects of testing, and curtail delays that are common in DW projects. Moreover, by using raw DV loads, keeping transformations to a minimum in the ETL process which loads the DV from the source, and allowing "Permissible Load Errors" in the DV, we can avoid the constant building up and tearing down associated with traditional staging. This raw DV load approach is appealing for highly regulated industries, such as banking and healthcare, with stricter standards for preservation of data (including erroneous ones) for high auditability, and compliance with mandates from both the private and public sectors. This strategy also fosters a more agile approach of getting DV up quickly, and incrementally building out the individual Data Marts. It must be noted however that by downplaying the ETL at the staging area of the data warehouse, the complex integration requirement is pushed downstream closer to the end user and must be carried out at that point.

The approach of raw direct DV loads is not only feasible, but is also very attractive due to the following interesting observation: the DV represents the enterprise data warehouse (EDW), but not all data in the DV will be relevant/interesting for loading into DMs further downstream for either reporting, analysis, or mining. As such, it would not be prudent to allow these data items to hold up a staging process unnecessarily as the DV is capable of allowing certain errors to be stored and corrected later.

Even though this work focuses on the relational model and transaction data for the sources, the load patterns used are flexible, reusable, and resumable. The approach presented is also general enough to make it applicable to other models and projects where a DV-based data warehouse can be implemented; additional research will have to be conducted to verify this conjecture. Hopefully this research opens up new avenues for exploration of the DV as a viable option for raw direct loads to simplify the staging process, while realizing the benefits that the DV-based approach provides.

REFERENCES

Collins, G., Hogan, M., Shibley, M., Williams, C., & Jovanovich, V. (2014). Data Vault and HQDM Principles, *Proceedings of the Southern Association for Information Systems, Paper #3*.

Corr, L. & Stagnitto, J. (2011). Agile Data Warehouse Design: Collaborative Dimensional Modeling, from Whiteboard to Star Schema (May 2013 revision), Decision One Press

Devlin, B., & Cote, L. D. (1996). *Data warehouse: from architecture to implementation*. Addison-Wesley Longman Publishing Co., Inc.

Generatedata (2015, August 22). Retrieve from http://www.generatedata.com.

Goldman, L. (2007). Data Warehouse Quality Assurance Best Practices. Retrieved from www.information-management.com/issues/20070801/1089385-1.html?.

Golfarelli, M., & Rizzi, S. (2009a). *Data Warehouse design: Modern principles and methodologies*. McGraw-Hill, Inc.

Golfarelli, M., & Rizzi, S. (2009b). A comprehensive approach to data warehouse testing. In *Proceedings of the ACM twelfth international workshop on Data warehousing and OLAP* (pp. 17-24). ACM.

Graziano, K. (2011). Introduction to Data Vault Modeling. *True BridgeResources, White paper*.

Hughes, R. (2015). *Agile Data Warehousing for the Enterprise: A Guide for Solution Architects and Project Leaders.* Morgan Kaufmann.

Inmon, W. H., and Linstedt, D. (2014). *Data Architecture: A Primer for the Data Scientist: Big Data, Data Warehouse and Data Vault*. Morgan Kaufmann.

Ivanova, M., Kersten, M., & Manegold, S. (2012). Data vaults: a symbiosis between database technology and scientific file repositories. *In Scientific and Statistical Database Management,* January 2012 (pp. 485-494). Springer Berlin Heidelberg.

Ivanova, M., Kargin, Y., Kersten, M., Manegold, S., Zhang, Y., Datcu, M., & Molina, D. E. (2013). Data vaults: a database welcome to scientific file repositories. In *Proceedings of the 25th International Conference on Scientific and Statistical Database Management,* July 2013 (p. 48). ACM.

Jovanovic, V., & Bojicic, I. (2012). Conceptual data vault model. In *SAIS Conference, Atlanta, Georgia: March* (Vol. 23, pp. 1-6).

Jovanovic, V., Bojicic, I., Knowles, C., Pavlic, M., & Informatike, O. (2012). Persistent staging area models for Data Warehouses. *Issues in Information Systems*, *13*(1), 121-132.

Kamal, R., & Nakul M. (2010). Adventures with Testing BI/DW Application: On a crusade to find the Holy Grail.
Retrieved from http://msdn.microsoft.com/en-us/library/gg248101.aspx.

Kimball, R., & Ross, M. (2013). *The data warehouse toolkit: The definitive guide to dimensional modeling*. John Wiley & Sons.

Linstedt, D., & Graziano, K. (2011). *Super Charge Your Data Warehouse: Invaluable Data Modeling Rules to Implement Your Data Vault*. CreateSpace.

Linstedt, D., & Olschimke, M. (2015). *Building a Scalable Data Warehouse with Data Vault 2.0: Implementation Guide for Microsoft SQL Server 2014*. Morgan Kaufmann.

Martinez-Rubi, O., Kersten, M. L., Goncalves, R., & Ivanova, M. (2014). A column-store meets the point clouds. *FOSS4G-Europe Academic Track*.

Mathen, M. P. (2010). Data Warehouse Testing. *Infosys White paper published in the DeveloperIQ Magazine, Year of Publication*.

Mundy, J. (2011). Design Tip #134 Data Warehouse Testing Recommendations (Kimball Group). Retrieved from http://www.kimballgroup.com/2011/05/04/design-tip-134-data-warehouse-testing-recommendations/.

Rahm, E., & Do, H. H. (2000). Data cleaning: Problems and current approaches. *IEEE Data Eng. Bull.*, *23*(4), 3-13.

Rao R. N., Ramesh K., & Jamuna R. N. (2009). Key Factors for an Effective Quality Assurance in Data Warehousing, "2 rao nemani, key factors for an effective qa in dw.pdf*", In Proceedings of the Silicon Valley American Society for Quality Conference.*

Rizzi, S. (2008). Data warehouse *In Encyclopedia of Computer Science and Engineering, B.W. Wah (Ed)*, John Wiley & Sons.

Vucevic, D., & Zhang, M. J. (2011). *Testing Data Warehouse Applications*. Trafford Publishing.

Yaddow, W. (2013). Data Warehouse Testing: Part 1 - Conducting end-to-end testing and quality assurance for data warehouses. Retrieved from http://ibmdatamag.com/2013/07/data-warehouse-testing-part-1/?.

APPENDIX A

SQL DATA DEFINITION LANGUAGE (DDL) SCRIPTS

Rental Source

```
USE [SourceRental]
GO


/****** Object:  Table [dbo].[AGENT]    ******/
IF EXISTS(SELECT 1 FROM sys.tables WHERE object_id = OBJECT_ID('AGENT'))
BEGIN
   DROP TABLE [dbo].[AGENT]
END
GO

/****** Object:  Table [dbo].[AGENT]   ******/
SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

SET ANSI_PADDING ON
GO

CREATE TABLE [dbo].[AGENT](
        [AgentId] [int] NOT NULL,
        [BusinessName] [varchar](64) NOT NULL,
        [Street] [varchar](64) NOT NULL,
        [City] [varchar](64) NOT NULL,
        [ZipCode] [varchar](10) NOT NULL,
        [State] [varchar](2) NOT NULL,
PRIMARY KEY CLUSTERED
(
        [AgentId] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]
```

```
GO

SET ANSI_PADDING OFF
GO


USE [SourceRental]
GO

/****** Object:  Table [dbo].[CUSTOMER]    ******/
IF EXISTS(SELECT 1 FROM sys.tables WHERE object_id = OBJECT_ID('CUSTOMER'))
BEGIN
   DROP TABLE [dbo].[CUSTOMER]
END
GO

/****** Object:  Table [dbo].[CUSTOMER]    ******/
SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

SET ANSI_PADDING ON
GO

CREATE TABLE [dbo].[CUSTOMER](
            [CustomerId] [int] NOT NULL,
            [LastName] [varchar](32) NOT NULL,
            [FirstName] [varchar](32) NOT NULL,
            [DateOfBirth] [date] NOT NULL,
            [Street] [varchar](64) NOT NULL,
            [City] [varchar](64) NOT NULL,
            [ZipCode] [varchar](10) NOT NULL,
            [State] [varchar](2) NOT NULL,
```

```
	[PhoneNumber] [varchar](20) NOT NULL,
PRIMARY KEY CLUSTERED
(
	[CustomerId] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]

GO

SET ANSI_PADDING OFF
GO



USE [SourceRental]
GO

/****** Object:  Table [dbo].[VEHICLE]    ******/
IF EXISTS(SELECT 1 FROM sys.tables WHERE object_id = OBJECT_ID('VEHICLE'))
BEGIN
   DROP TABLE [dbo].[VEHICLE]
END
GO

/****** Object:  Table [dbo].[VEHICLE]    ******/
SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

SET ANSI_PADDING ON
GO

CREATE TABLE [dbo].[VEHICLE](
	[VehicleId] [int] NOT NULL,
	[Make] [varchar](64) NOT NULL,
```

```
	[Model] [varchar](64) NOT NULL,
	[Year] [int] NOT NULL,
	[Color] [varchar](32) NOT NULL,
PRIMARY KEY CLUSTERED
(
	[VehicleId] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]

GO

SET ANSI_PADDING OFF
GO



USE [SourceRental]
GO

/****** Object:  Table [dbo].[RENTAL]    ******/
IF EXISTS(SELECT 1 FROM sys.tables WHERE object_id = OBJECT_ID('RENTAL'))
BEGIN
   DROP TABLE dbo.[RENTAL]
END
GO

/****** Object:  Table [dbo].[RENTAL]    ******/
SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

CREATE TABLE [dbo].[RENTAL](
	[RentalNo] [varchar](15) NOT NULL,
	[VehicleId] [int] NOT NULL,
	[AgentId] [int] NOT NULL,
```

```
	[CustomerId] [int] NOT NULL,
	[RentalDate] [date] NOT NULL,
	[NumDays] [int] NOT NULL,
	[Cost] [money] NOT NULL,
PRIMARY KEY CLUSTERED
(
	[RentalNo] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]


GO
```

Data Vault 1.0

```
USE [DV1Rental]
GO

IF EXISTS(SELECT 1 FROM sys.tables WHERE object_id = OBJECT_ID('Hub_Agent'))
BEGIN
	ALTER TABLE [dbo].[Hub_Agent] DROP CONSTRAINT [DF_Hub_Agent_Sur_Seq]
END
GO

/****** Object:  Table [dbo].[Hub_Agent]    ******/
IF EXISTS(SELECT 1 FROM sys.tables WHERE object_id = OBJECT_ID('Hub_Agent'))
BEGIN
	DROP TABLE [dbo].[Hub_Agent]
END
GO


IF EXISTS(SELECT 1 FROM sys.tables WHERE object_id = OBJECT_ID('Hub_Customer'))
BEGIN
	ALTER TABLE [dbo].[Hub_Customer] DROP CONSTRAINT [DF_Hub_Customer_Sur_Seq]
END
GO

/****** Object:  Table [dbo].[Hub_Customer]    ******/
IF EXISTS(SELECT 1 FROM sys.tables WHERE object_id = OBJECT_ID('Hub_Customer'))
BEGIN
	DROP TABLE [dbo].[Hub_Customer]
END
GO


IF EXISTS(SELECT 1 FROM sys.tables WHERE object_id = OBJECT_ID('Hub_Vehicle'))
BEGIN
```

```
	ALTER TABLE [dbo].[Hub_Vehicle] DROP CONSTRAINT [DF_Hub_Vehicle_Sur_Seq]
END
GO

/****** Object:  Table [dbo].[Hub_Vehicle]    ******/
IF EXISTS(SELECT 1 FROM sys.tables WHERE object_id = OBJECT_ID('Hub_Vehicle'))
BEGIN
	DROP TABLE [dbo].[Hub_Vehicle]
END
GO


IF EXISTS(SELECT 1 FROM sys.tables WHERE object_id = OBJECT_ID('Link_Rental'))
BEGIN
	ALTER TABLE [dbo].[Link_Rental] DROP CONSTRAINT [DF_Link_Rental_Sur_Seq]
END
GO


/****** Object:  Table [dbo].[Link_Rental]    ******/

IF EXISTS(SELECT 1 FROM sys.tables WHERE object_id = OBJECT_ID('Link_Rental'))
BEGIN
	DROP TABLE [dbo].[Link_Rental]
END
GO


/****** Object:  Table [dbo].[Sat_Agent]    ******/
IF EXISTS(SELECT 1 FROM sys.tables WHERE object_id = OBJECT_ID('Sat_Agent'))
BEGIN
	DROP TABLE [dbo].[Sat_Agent]
END
GO


/****** Object:  Table [dbo].[Sat_Customer]    ******/
IF EXISTS(SELECT 1 FROM sys.tables WHERE object_id = OBJECT_ID('Sat_Customer'))
BEGIN
	DROP TABLE [dbo].[Sat_Customer]
END
GO


/****** Object:  Table [dbo].[Sat_Rental]    ******/
IF EXISTS(SELECT 1 FROM sys.tables WHERE object_id = OBJECT_ID('Sat_Rental'))
BEGIN
	DROP TABLE [dbo].[Sat_Rental]
END
GO


/****** Object:  Table [dbo].[Sat_Vehicle]    ******/
IF EXISTS(SELECT 1 FROM sys.tables WHERE object_id = OBJECT_ID('Sat_Vehicle'))
BEGIN
	DROP TABLE [dbo].[Sat_Vehicle]
```

```
END
GO

USE [DV1Rental]
GO

/****** Object:  Sequence [dbo].[AGENT_SUR_SEQ]    ******/
DROP SEQUENCE [dbo].[AGENT_SUR_SEQ]
GO

USE [DV1Rental]
GO

/****** Object:  Sequence [dbo].[AGENT_SUR_SEQ]    ******/
CREATE SEQUENCE [dbo].[AGENT_SUR_SEQ]
 AS [bigint]
 START WITH 1
 INCREMENT BY 1
 MINVALUE -9223372036854775808
 MAXVALUE 9223372036854775807
 CACHE  10
GO


USE [DV1Rental]
GO

/****** Object:  Sequence [dbo].[CUSTOMER_SUR_SEQ]    ******/
DROP SEQUENCE [dbo].[CUSTOMER_SUR_SEQ]
GO

USE [DV1Rental]
GO

/****** Object:  Sequence [dbo].[CUSTOMER_SUR_SEQ]    ******/
CREATE SEQUENCE [dbo].[CUSTOMER_SUR_SEQ]
 AS [bigint]
 START WITH 1
 INCREMENT BY 1
 MINVALUE -9223372036854775808
 MAXVALUE 9223372036854775807
 CACHE  10
GO


USE [DV1Rental]
GO

/****** Object:  Sequence [dbo].[VEHICLE_SUR_SEQ]    ******/
DROP SEQUENCE [dbo].[VEHICLE_SUR_SEQ]
GO

USE [DV1Rental]
GO

/****** Object:  Sequence [dbo].[VEHICLE_SUR_SEQ]    ******/
```

```
CREATE SEQUENCE [dbo].[VEHICLE_SUR_SEQ]
 AS [bigint]
 START WITH 1
 INCREMENT BY 1
 MINVALUE -9223372036854775808
 MAXVALUE 9223372036854775807
 CACHE  10
GO


USE [DV1Rental]
GO

/****** Object:  Sequence [dbo].[RENTAL_SUR_SEQ]    ******/
DROP SEQUENCE [dbo].[RENTAL_SUR_SEQ]
GO

USE [DV1Rental]
GO

/****** Object:  Sequence [dbo].[RENTAL_SUR_SEQ]    ******/
CREATE SEQUENCE [dbo].[RENTAL_SUR_SEQ]
 AS [bigint]
 START WITH 1
 INCREMENT BY 1
 MINVALUE -9223372036854775808
 MAXVALUE 9223372036854775807
 CACHE  10
GO

USE [DV1Rental]
GO

IF EXISTS(SELECT 1 FROM sys.tables WHERE object_id = OBJECT_ID('Hub_Agent'))
BEGIN
	ALTER TABLE [dbo].[Hub_Agent] DROP CONSTRAINT [DF_Hub_Agent_Sur_Seq]
END
GO

/****** Object:  Table [dbo].[Hub_Agent]    ******/
IF EXISTS(SELECT 1 FROM sys.tables WHERE object_id = OBJECT_ID('Hub_Agent'))
BEGIN
	DROP TABLE [dbo].[Hub_Agent]
END
GO

/****** Object:  Table [dbo].[Hub_Agent]    ******/
SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

CREATE TABLE [dbo].[Hub_Agent](
	[Agent_Seq] [bigint] NOT NULL,
	[Agent_Bk] [int] NOT NULL,
```

```
	[LoadTimestamp] [datetime] NOT NULL,
	[LoadProcess] [bigint] NOT NULL,
	[RecordSource] [nvarchar](128) NOT NULL,
 CONSTRAINT [PK_Hub_Agent] PRIMARY KEY CLUSTERED
(
	[Agent_Seq] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]

GO

ALTER TABLE [dbo].[Hub_Agent] ADD CONSTRAINT [DF_Hub_Agent_Sur_Seq] DEFAULT (NEXT VALUE
FOR [dbo].[AGENT_SUR_SEQ]) FOR [Agent_Seq]
GO
USE [DV1Rental]
GO

IF EXISTS(SELECT 1 FROM sys.tables WHERE object_id = OBJECT_ID('Hub_Customer'))
BEGIN
	ALTER TABLE [dbo].[Hub_Customer] DROP CONSTRAINT [DF_Hub_Customer_Sur_Seq]
END
GO

/****** Object:  Table [dbo].[Hub_Customer]    ******/
IF EXISTS(SELECT 1 FROM sys.tables WHERE object_id = OBJECT_ID('Hub_Customer'))
BEGIN
	DROP TABLE [dbo].[Hub_Customer]
END
GO

/****** Object:  Table [dbo].[Hub_Customer]    ******/
SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

CREATE TABLE [dbo].[Hub_Customer](
	[Customer_Seq] [bigint] NOT NULL,
	[Customer_Bk] [int] NOT NULL,
	[LoadTimestamp] [datetime] NOT NULL,
	[LoadProcess] [bigint] NOT NULL,
	[RecordSource] [nvarchar](128) NOT NULL,
 CONSTRAINT [PK_Hub_Customer] PRIMARY KEY CLUSTERED
(
	[Customer_Seq] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]

GO

ALTER TABLE [dbo].[Hub_Customer] ADD CONSTRAINT [DF_Hub_Customer_Sur_Seq] DEFAULT (NEXT
VALUE FOR [dbo].[CUSTOMER_SUR_SEQ]) FOR [Customer_Seq]
GO
```

```
USE [DV1Rental]
GO

IF EXISTS(SELECT 1 FROM sys.tables WHERE object_id = OBJECT_ID('Hub_Vehicle'))
BEGIN
	ALTER TABLE [dbo].[Hub_Vehicle] DROP CONSTRAINT [DF_Hub_Vehicle_Sur_Seq]
END
GO

/****** Object:  Table [dbo].[Hub_Vehicle]    ******/
IF EXISTS(SELECT 1 FROM sys.tables WHERE object_id = OBJECT_ID('Hub_Vehicle'))
BEGIN
	DROP TABLE [dbo].[Hub_Vehicle]
END
GO

/****** Object:  Table [dbo].[Hub_Vehicle]    ******/
SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

CREATE TABLE [dbo].[Hub_Vehicle](
	[Vehicle_Seq] [bigint] NOT NULL,
	[Vehicle_Bk] [int] NOT NULL,
	[LoadTimestamp] [datetime] NOT NULL,
	[LoadProcess] [bigint] NOT NULL,
	[RecordSource] [nvarchar](128) NOT NULL,
 CONSTRAINT [PK_Hub_Vehicle] PRIMARY KEY CLUSTERED
(
	[Vehicle_Seq] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]

GO

ALTER TABLE [dbo].[Hub_Vehicle] ADD CONSTRAINT [DF_Hub_Vehicle_Sur_Seq] DEFAULT (NEXT
VALUE FOR [dbo].[VEHICLE_SUR_SEQ]) FOR [Vehicle_Seq]
GO
USE [DV1Rental]
GO

IF EXISTS(SELECT 1 FROM sys.tables WHERE object_id = OBJECT_ID('Link_Rental'))
BEGIN
	ALTER TABLE [dbo].[Link_Rental] DROP CONSTRAINT [DF_Link_Rental_Sur_Seq]
END
GO


/****** Object:  Table [dbo].[Link_Rental]    ******/

IF EXISTS(SELECT 1 FROM sys.tables WHERE object_id = OBJECT_ID('Link_Rental'))
BEGIN
	DROP TABLE [dbo].[Link_Rental]
```

```
END
GO

/****** Object:  Table [dbo].[Link_Rental]    ******/
SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

CREATE TABLE [dbo].[Link_Rental](
	[Rental_Seq] [bigint] NOT NULL,
	[Customer_Seq] [bigint] NOT NULL,
	[Agent_Seq] [bigint] NOT NULL,
	[Vehicle_Seq] [bigint] NOT NULL,
	[LoadTimestamp] [datetime] NOT NULL,
	[LoadProcess] [bigint] NOT NULL,
	[RecordSource] [nvarchar](128) NOT NULL,
 CONSTRAINT [PK_Link_Rental] PRIMARY KEY CLUSTERED
(
	[Rental_Seq] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]

GO

ALTER TABLE [dbo].[Link_Rental] ADD  CONSTRAINT [DF_Link_Rental_Sur_Seq]  DEFAULT (NEXT
VALUE FOR [dbo].[RENTAL_SUR_SEQ]) FOR [Rental_Seq]
GO

USE [DV1Rental]
GO

/****** Object:  Table [dbo].[Sat_Agent]    ******/
IF EXISTS(SELECT 1 FROM sys.tables WHERE object_id = OBJECT_ID('Sat_Agent'))
BEGIN
	DROP TABLE [dbo].[Sat_Agent]
END
GO

/****** Object:  Table [dbo].[Sat_Agent]    ******/
SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

CREATE TABLE [dbo].[Sat_Agent](
	[Agent_Seq] [bigint] NOT NULL,
	[Valid_From_Date] [datetime] NOT NULL,
	[Valid_To_Date] [date] NULL,
	[BusinessName] [varchar](64) NOT NULL,
	[Street] [varchar](64) NOT NULL,
	[City] [varchar](64) NOT NULL,
	[ZipCode] [varchar](10) NOT NULL,
```

```
	[State] [varchar](2) NOT NULL,
	[LoadTimestamp] [datetime] NOT NULL,
	[LoadProcess] [bigint] NOT NULL,
	[RecordSource] [nvarchar](128) NOT NULL,
 CONSTRAINT [PK_Sat_Agent] PRIMARY KEY CLUSTERED
(
	[Agent_Seq] ASC,
	[Valid_From_Date] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]

GO
USE [DV1Rental]
GO

/****** Object:  Table [dbo].[Sat_Customer]    ******/
IF EXISTS(SELECT 1 FROM sys.tables WHERE object_id = OBJECT_ID('Sat_Customer'))
BEGIN
	DROP TABLE [dbo].[Sat_Customer]
END
GO

/****** Object:  Table [dbo].[Sat_Customer]    ******/
SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

CREATE TABLE [dbo].[Sat_Customer](
	[Customer_Seq] [bigint] NOT NULL,
	[Valid_From_Date] [datetime] NOT NULL,
	[Valid_To_Date] [date] NULL,
	[LastName] [varchar](32) NOT NULL,
	[FirstName] [varchar](32) NOT NULL,
	[DateOfBirth] [date] NOT NULL,
	[Street] [varchar](64) NOT NULL,
	[City] [varchar](64) NOT NULL,
	[ZipCode] [varchar](10) NOT NULL,
	[State] [varchar](2) NOT NULL,
	[PhoneNumber] [varchar](20) NOT NULL,
	[LoadTimestamp] [datetime] NOT NULL,
	[LoadProcess] [bigint] NOT NULL,
	[RecordSource] [nvarchar](128) NOT NULL,
 CONSTRAINT [PK_Sat_Customer] PRIMARY KEY CLUSTERED
(
	[Customer_Seq] ASC,
	[Valid_From_Date] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]

GO

USE [DV1Rental]
```

```
GO

/****** Object:  Table [dbo].[Sat_Vehicle]    ******/
IF EXISTS(SELECT 1 FROM sys.tables WHERE object_id = OBJECT_ID('Sat_Vehicle'))
BEGIN
	DROP TABLE [dbo].[Sat_Vehicle]
END
GO

/****** Object:  Table [dbo].[Sat_Vehicle]    ******/
SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

CREATE TABLE [dbo].[Sat_Vehicle](
	[Vehicle_Seq] [bigint] NOT NULL,
	[Valid_From_Date] [datetime] NOT NULL,
	[Valid_To_Date] [date] NULL,
	[Make] [varchar](64) NOT NULL,
	[Model] [varchar](64) NOT NULL,
	[Year] [int] NOT NULL,
	[Color] [varchar](32) NOT NULL,
	[LoadTimestamp] [datetime] NOT NULL,
	[LoadProcess] [bigint] NOT NULL,
	[RecordSource] [nvarchar](128) NOT NULL,
 CONSTRAINT [PK_Sat_Vehicle] PRIMARY KEY CLUSTERED
(
	[Vehicle_Seq] ASC,
	[Valid_From_Date] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]

GO
USE [DV1Rental]
GO

/****** Object:  Table [dbo].[Sat_Rental]    ******/
IF EXISTS(SELECT 1 FROM sys.tables WHERE object_id = OBJECT_ID('Sat_Rental'))
BEGIN
	DROP TABLE [dbo].[Sat_Rental]
END
GO

/****** Object:  Table [dbo].[Sat_Rental]    ******/
SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

CREATE TABLE [dbo].[Sat_Rental](
	[Rental_Seq] [bigint] NOT NULL,
	[Valid_From_Date] [datetime] NOT NULL,
```

```
	[Valid_To_Date] [date] NULL,
	[RentalNo] [varchar](15) NOT NULL,
	[RentalDate] [date] NOT NULL,
	[NumDays] [int] NOT NULL,
	[Cost] [money] NOT NULL,
	[LoadTimestamp] [datetime] NOT NULL,
	[LoadProcess] [bigint] NOT NULL,
	[RecordSource] [nvarchar](128) NOT NULL,
 CONSTRAINT [PK_Sat_Rental] PRIMARY KEY CLUSTERED
(
	[Rental_Seq] ASC,
	[Valid_From_Date] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]

GO
```

Data Vault 2.0

```
USE [DV2Rental]
GO


/****** Object:  Table [dbo].[Hub_Agent]    ******/
IF EXISTS(SELECT 1 FROM sys.tables WHERE object_id = OBJECT_ID('Hub_Agent'))
BEGIN
	DROP TABLE [dbo].[Hub_Agent]
END
GO


/****** Object:  Table [dbo].[Hub_Agent]    ******/
SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

CREATE TABLE [dbo].[Hub_Agent](
	[Agent_Hsk] [char](32) NOT NULL,
	[Agent_Bk] [int] NOT NULL,
	[LoadTimestamp] [datetime] NOT NULL,
	[LoadProcess] [bigint] NOT NULL,
	[RecordSource] [nvarchar](128) NOT NULL,
```

```
 CONSTRAINT [PK_Hub_Agent] PRIMARY KEY CLUSTERED
(
	[Agent_Hsk] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]

GO
USE [DV2Rental]
GO

/****** Object:  Table [dbo].[Hub_Customer]    ******/
IF EXISTS(SELECT 1 FROM sys.tables WHERE object_id = OBJECT_ID('Hub_Customer'))
BEGIN
	DROP TABLE [dbo].[Hub_Customer]
END
GO

/****** Object:  Table [dbo].[Hub_Customer]    ******/
SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

CREATE TABLE [dbo].[Hub_Customer](
	[Customer_Hsk] [char](32) NOT NULL,
	[Customer_Bk] [int] NOT NULL,
	[LoadTimestamp] [datetime] NOT NULL,
	[LoadProcess] [bigint] NOT NULL,
	[RecordSource] [nvarchar](128) NOT NULL,
 CONSTRAINT [PK_Hub_Customer] PRIMARY KEY CLUSTERED
(
	[Customer_Hsk] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]
```

```
GO
USE [DV2Rental]
GO

/****** Object:  Table [dbo].[Hub_Vehicle]    ******/
IF EXISTS(SELECT 1 FROM sys.tables WHERE object_id = OBJECT_ID('Hub_Vehicle'))
BEGIN
	DROP TABLE [dbo].[Hub_Vehicle]
END
GO

/****** Object:  Table [dbo].[Hub_Vehicle]    ******/
SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

CREATE TABLE [dbo].[Hub_Vehicle](
	[Vehicle_Hsk] [char](32) NOT NULL,
	[Vehicle_Bk] [int] NOT NULL,
	[LoadTimestamp] [datetime] NOT NULL,
	[LoadProcess] [bigint] NOT NULL,
	[RecordSource] [nvarchar](128) NOT NULL,
 CONSTRAINT [PK_Hub_Vehicle] PRIMARY KEY CLUSTERED
(
	[Vehicle_Hsk] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]

GO
USE [DV2Rental]
GO
```

```
/****** Object:  Table [dbo].[Link_Rental]    ******/
IF EXISTS(SELECT 1 FROM sys.tables WHERE object_id = OBJECT_ID('Link_Rental'))
BEGIN
	DROP TABLE [dbo].[Link_Rental]
END
GO

/****** Object:  Table [dbo].[Link_Rental]    ******/
SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

CREATE TABLE [dbo].[Link_Rental](
	[Rental_Hsk] [char](32) NOT NULL,
	[Customer_Hsk] [char](32) NOT NULL,
	[Agent_Hsk] [char](32) NOT NULL,
	[Vehicle_Hsk] [char](32) NOT NULL,
	[LoadTimestamp] [datetime] NOT NULL,
	[LoadProcess] [bigint] NOT NULL,
	[RecordSource] [nvarchar](128) NOT NULL,
 CONSTRAINT [PK_Link_Rental] PRIMARY KEY CLUSTERED
(
	[Rental_Hsk] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]

GO
USE [DV2Rental]
GO

/****** Object:  Table [dbo].[Sat_Agent]    ******/
IF EXISTS(SELECT 1 FROM sys.tables WHERE object_id = OBJECT_ID('Sat_Agent'))
BEGIN
	DROP TABLE [dbo].[Sat_Agent]
```

```
END
GO

/****** Object:  Table [dbo].[Sat_Agent]    ******/
SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

CREATE TABLE [dbo].[Sat_Agent](
	[Agent_Hsk] [char](32) NOT NULL,
	[Valid_From_Date] [datetime] NOT NULL,
	[Valid_To_Date] [date] NULL,
	[BusinessName] [varchar](64) NOT NULL,
	[Street] [varchar](64) NOT NULL,
	[City] [varchar](64) NOT NULL,
	[ZipCode] [varchar](10) NOT NULL,
	[State] [varchar](2) NOT NULL,
	[LoadTimestamp] [datetime] NOT NULL,
	[LoadProcess] [bigint] NOT NULL,
	[RecordSource] [nvarchar](128) NOT NULL,
 CONSTRAINT [PK_Sat_Agent] PRIMARY KEY CLUSTERED
(
	[Agent_Hsk] ASC,
	[Valid_From_Date] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]

GO
USE [DV2Rental]
GO

/****** Object:  Table [dbo].[Sat_Customer]    ******/
IF EXISTS(SELECT 1 FROM sys.tables WHERE object_id = OBJECT_ID('Sat_Customer'))
BEGIN
```

```
        DROP TABLE [dbo].[Sat_Customer]
END
GO

/****** Object:  Table [dbo].[Sat_Customer]    ******/
SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

CREATE TABLE [dbo].[Sat_Customer](
        [Customer_Hsk] [char](32) NOT NULL,
        [Valid_From_Date] [datetime] NOT NULL,
        [Valid_To_Date] [date] NULL,
        [LastName] [varchar](32) NOT NULL,
        [FirstName] [varchar](32) NOT NULL,
        [DateOfBirth] [date] NOT NULL,
        [Street] [varchar](64) NOT NULL,
        [City] [varchar](64) NOT NULL,
        [ZipCode] [varchar](10) NOT NULL,
        [State] [varchar](2) NOT NULL,
        [PhoneNumber] [varchar](20) NOT NULL,
        [LoadTimestamp] [datetime] NOT NULL,
        [LoadProcess] [bigint] NOT NULL,
        [RecordSource] [nvarchar](128) NOT NULL,
 CONSTRAINT [PK_Sat_Customer] PRIMARY KEY CLUSTERED
(
        [Customer_Hsk] ASC,
        [Valid_From_Date] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]

GO
USE [DV2Rental]
GO
```

```
/****** Object:  Table [dbo].[Sat_Vehicle]    ******/
IF EXISTS(SELECT 1 FROM sys.tables WHERE object_id = OBJECT_ID('Sat_Vehicle'))
BEGIN
	DROP TABLE [dbo].[Sat_Vehicle]
END
GO

/****** Object:  Table [dbo].[Sat_Vehicle]    ******/
SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

CREATE TABLE [dbo].[Sat_Vehicle](
	[Vehicle_Hsk] [char](32) NOT NULL,
	[Valid_From_Date] [datetime] NOT NULL,
	[Valid_To_Date] [date] NULL,
	[Make] [varchar](64) NOT NULL,
	[Model] [varchar](64) NOT NULL,
	[Year] [int] NOT NULL,
	[Color] [varchar](32) NOT NULL,
	[LoadTimestamp] [datetime] NOT NULL,
	[LoadProcess] [bigint] NOT NULL,
	[RecordSource] [nvarchar](128) NOT NULL,
 CONSTRAINT [PK_Sat_Vehicle] PRIMARY KEY CLUSTERED
(
	[Vehicle_Hsk] ASC,
	[Valid_From_Date] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]

GO
USE [DV2Rental]
GO
```

```
/****** Object:  Table [dbo].[Sat_Rental]    ******/
IF EXISTS(SELECT 1 FROM sys.tables WHERE object_id = OBJECT_ID('Sat_Rental'))
BEGIN
	DROP TABLE [dbo].[Sat_Rental]
END
GO

/****** Object:  Table [dbo].[Sat_Rental]    ******/
SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

CREATE TABLE [dbo].[Sat_Rental](
	[Rental_Hsk] [char](32) NOT NULL,
	[Valid_From_Date] [datetime] NOT NULL,
	[Valid_To_Date] [date] NULL,
	[RentalNo] [varchar](15) NOT NULL,
	[RentalDate] [date] NOT NULL,
	[NumDays] [int] NOT NULL,
	[Cost] [money] NOT NULL,
	[LoadTimestamp] [datetime] NOT NULL,
	[LoadProcess] [bigint] NOT NULL,
	[RecordSource] [nvarchar](128) NOT NULL,
 CONSTRAINT [PK_Sat_Rental] PRIMARY KEY CLUSTERED
(
	[Rental_Hsk] ASC,
	[Valid_From_Date] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]

GO
```

Data Mart

```
USE [MartRental]
GO

IF OBJECT_ID(N'dbo.FT_RENTAL', N'U') IS NOT NULL
BEGIN
	DROP TABLE [dbo].[FT_RENTAL]
END


IF OBJECT_ID(N'dbo.DT_AGENT', N'U') IS NOT NULL
BEGIN
	DROP TABLE [dbo].[DT_AGENT]
END
GO


IF OBJECT_ID(N'dbo.DT_CUSTOMER', N'U') IS NOT NULL
BEGIN
	DROP TABLE [dbo].[DT_CUSTOMER]
END
GO


IF OBJECT_ID(N'dbo.DT_VEHICLE', N'U') IS NOT NULL
BEGIN
	DROP TABLE [dbo].[DT_VEHICLE]
END
GO
USE [MartRental]
GO

/****** Object:  Table [dbo].[DT_AGENT]    ******/
IF EXISTS(SELECT 1 FROM sys.tables WHERE object_id = OBJECT_ID('DT_AGENT'))
BEGIN
   DROP TABLE [dbo].[DT_AGENT]
END
GO

/****** Object:  Table [dbo].[DT_AGENT]    ******/
SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

SET ANSI_PADDING ON
GO

CREATE TABLE [dbo].[DT_AGENT](
```

```
	[Agent_ID] [int] NOT NULL,

	[BusinessName] [varchar](64) NOT NULL,
	[City] [varchar](64) NOT NULL,
	[ZipCode] [varchar](10) NOT NULL,
	[State] [varchar](2) NOT NULL,

	-- SCD Type 2 Audit Fields
	--[Is_Current_Record]	[bit]			NOT NULL DEFAULT 1,
	[Record_Start_Date]		[datetime]		NOT NULL DEFAULT GETDATE(),
	[Record_End_Date]		[date]			NULL,

	-- ETL Audit Fields
	[ETL_Load_Job]			[bigint]		NOT NULL DEFAULT 0,
	[Last_Load_Date]		[datetime]		NOT NULL DEFAULT GETDATE(),
	--[Last_Update_Date]	[datetime]		NOT NULL DEFAULT GETDATE(),
	[RecordSource]			[nvarchar](128) NOT NULL,

PRIMARY KEY CLUSTERED
(
	[Agent_ID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]

GO

SET ANSI_PADDING OFF
GO
USE [MartRental]
GO

/****** Object:  Table [dbo].[DT_CUSTOMER]    ******/
IF EXISTS(SELECT 1 FROM sys.tables WHERE object_id = OBJECT_ID('DT_CUSTOMER'))
BEGIN
   DROP TABLE [dbo].[DT_CUSTOMER]
END
GO

/****** Object:  Table [dbo].[DT_CUSTOMER]    ******/
SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

SET ANSI_PADDING ON
GO

CREATE TABLE [dbo].[DT_CUSTOMER](
	[Customer_ID] [int] NOT NULL,

	[Name] [varchar](65) NOT NULL,
	[DateOfBirth] [date] NOT NULL,
	[City] [varchar](64) NOT NULL,
	[ZipCode] [varchar](10) NOT NULL,
```

```
	[State] [varchar](2) NOT NULL,

	-- SCD Type 2 Audit Fields
	-- [Is_Current_Record]	[bit]			NOT NULL DEFAULT 1,
	[Record_Start_Date]		[date]			NOT NULL DEFAULT GETDATE(),
	[Record_End_Date]		[date]			NULL,

	-- ETL Audit Fields
	[ETL_Load_Job]			[bigint]		NOT NULL DEFAULT 0,
	[Last_Load_Date]		[datetime]		NOT NULL DEFAULT GETDATE(),
	--[Last_Update_Date]	[datetime]		NOT NULL DEFAULT GETDATE(),
	[RecordSource]			[nvarchar](128) NOT NULL,

PRIMARY KEY CLUSTERED
(
	[Customer_ID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]

GO

SET ANSI_PADDING OFF
GO
USE [MartRental]
GO

/****** Object:  Table [dbo].[DT_VEHICLE]    ******/
IF EXISTS(SELECT 1 FROM sys.tables WHERE object_id = OBJECT_ID('DT_VEHICLE'))
BEGIN
   DROP TABLE [dbo].[DT_VEHICLE]
END
GO

/****** Object:  Table [dbo].[DT_VEHICLE]    ******/
SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

SET ANSI_PADDING ON
GO

CREATE TABLE [dbo].[DT_VEHICLE](
	[Vehicle_ID] [int] NOT NULL,

	[Make] [varchar](64) NOT NULL,
	[Model] [varchar](64) NOT NULL,
	[Year] [int] NOT NULL,
	[Color] [varchar](32) NOT NULL,

	-- SCD Type 2 Audit Fields
	--[Is_Current_Record]	[bit]			NOT NULL DEFAULT 1,
	[Record_Start_Date]		[datetime]		NOT NULL DEFAULT GETDATE(),
	[Record_End_Date]		[date]			NULL,
```

```
	-- ETL Audit Fields
	[ETL_Load_Job]		[bigint]		NOT NULL DEFAULT 0,
	[Last_Load_Date]		[datetime]			NOT NULL DEFAULT GETDATE(),
	--[Last_Update_Date]	[datetime]			NOT NULL DEFAULT GETDATE(),
	[RecordSource]		[nvarchar](128) NOT NULL,

PRIMARY KEY CLUSTERED
(
	[Vehicle_ID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]

GO

SET ANSI_PADDING OFF
GO
USE [MartRental]
GO

/****** Object:  Table [dbo].[FT_RENTAL]    ******/
IF EXISTS(SELECT 1 FROM sys.tables WHERE object_id = OBJECT_ID('FT_RENTAL'))
BEGIN
   DROP TABLE dbo.[FT_RENTAL]
END
GO

/****** Object:  Table [dbo].[FT_RENTAL]    ******/
SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

CREATE TABLE [dbo].[FT_RENTAL](
	[Rental_Date_ID] BIGINT NOT NULL,
	[Agent_ID] [int] NOT NULL,
	[Customer_ID] [int] NOT NULL,
	[Vehicle_ID] [int] NOT NULL,

	[NumDays] [int] NOT NULL,
	[Cost] [money] NOT NULL,
PRIMARY KEY CLUSTERED
(
	[Rental_Date_ID],
	[Vehicle_ID],
	[Agent_ID],
	[Customer_ID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]

GO

-- Foreign key constraints
```

```
ALTER TABLE [dbo].[FT_RENTAL] ADD  CONSTRAINT [FK_Rental_FT_Date]  FOREIGN KEY
(Rental_Date_ID) REFERENCES DT_DATE(DateKey)
GO
ALTER TABLE [dbo].[FT_RENTAL] ADD  CONSTRAINT [FK_Rental_FT_Agent]  FOREIGN KEY
(Agent_ID) REFERENCES DT_AGENT(Agent_ID)
GO
ALTER TABLE [dbo].[FT_RENTAL] ADD  CONSTRAINT [FK_Rental_FT_Customer]  FOREIGN KEY
(Customer_ID) REFERENCES DT_CUSTOMER(Customer_ID)
GO
ALTER TABLE [dbo].[FT_RENTAL] ADD  CONSTRAINT [FK_Rental_FT_Vehicle]  FOREIGN KEY
(Vehicle_ID) REFERENCES DT_VEHICLE(Vehicle_ID)
GO



/*
        The Date dimension table below is based on the Kimball Date dimension.
        It is modified slightly for the purpose of this work
*/

USE [MartRental]
GO

/* Drop DT_Date dimension table */
IF EXISTS ( SELECT  *
        FROM    dbo.sysobjects
        WHERE   id = OBJECT_ID(N'dbo.DT_Date')
            AND OBJECTPROPERTY(id, N'IsUserTable') = 1 )
  DROP TABLE [dbo].[DT_Date]
GO

/* Create table DimDate */
CREATE TABLE [dbo].[DT_Date]
    ( [DateKey] BIGINT NOT NULL
    , [FullDate] DATETIME NULL
    , [DateName] CHAR(11) NULL
    , [DayOfWeek] TINYINT NULL
    , [DayNameOfWeek] CHAR(10) NULL
    , [DayOfMonth] TINYINT NULL
    , [DayOfYear] SMALLINT NULL
    , [WeekdayWeekend] CHAR(7) NULL
    , [WeekOfYear] TINYINT NULL
    , [MonthName] CHAR(10) NULL
    , [MonthOfYear] TINYINT NULL
    , [IsLastDayOfMonth] CHAR(1) NULL
    , [CalendarQuarter] TINYINT NULL
    , [CalendarYear] SMALLINT NULL
    , [CalendarYearMonth] CHAR(7) NULL
    , [CalendarYearQtr] CHAR(7) NULL
    , [FiscalMonthOfYear] TINYINT NULL
    , [FiscalQuarter] TINYINT NULL
    , [FiscalYear] INT NULL
    , [FiscalYearMonth] CHAR(9) NULL
    , [FiscalYearQtr] CHAR(8) NULL
    , [AuditKey] BIGINT IDENTITY NOT NULL
    , CONSTRAINT [PK_DT_Date] PRIMARY KEY CLUSTERED ( [DateKey] )
```

```
    )
ON    [PRIMARY]
GO


-- Add extended properties for the DT_Date dimension table

EXEC sys.sp_addextendedproperty @name = N'Table Type', @value = N'Dimension',
  @level0type = N'SCHEMA', @level0name = N'dbo', @level1type = N'TABLE',
  @level1name = N'DT_Date';
EXEC sys.sp_addextendedproperty @name = N'Description',
  @value = N'Date dimension table stores one record for each day between the start and end date specified during
table loading',
  @level0type = N'SCHEMA', @level0name = N'dbo', @level1type = N'TABLE',
  @level1name = N'DT_Date';
GO

EXEC sys.sp_addextendedproperty @name = N'Description',
  @value = N'Surrogate primary key', @level0type = N'SCHEMA',
  @level0name = N'dbo', @level1type = N'TABLE', @level1name = N'DT_Date',
  @level2type = N'COLUMN', @level2name = N'DateKey' ;
EXEC sys.sp_addextendedproperty @name = N'Description',
  @value = N'Full date as a SQL date (time=00:00:00)', @level0type = N'SCHEMA',
  @level0name = N'dbo', @level1type = N'TABLE', @level1name = N'DT_Date',
  @level2type = N'COLUMN', @level2name = N'FullDate' ;
EXEC sys.sp_addextendedproperty @name = N'Description',
  @value = N'Standard Date Format of YYYY/MM/DD', @level0type = N'SCHEMA',
  @level0name = N'dbo', @level1type = N'TABLE', @level1name = N'DT_Date',
  @level2type = N'COLUMN', @level2name = N'DateName' ;
EXEC sys.sp_addextendedproperty @name = N'Description',
  @value = N'Number of the day of week; Sunday = 1', @level0type = N'SCHEMA',
  @level0name = N'dbo', @level1type = N'TABLE', @level1name = N'DT_Date',
  @level2type = N'COLUMN', @level2name = N'DayOfWeek' ;
EXEC sys.sp_addextendedproperty @name = N'Description',
  @value = N'Day name of week', @level0type = N'SCHEMA', @level0name = N'dbo',
  @level1type = N'TABLE', @level1name = N'DT_Date', @level2type = N'COLUMN',
  @level2name = N'DayNameOfWeek' ;
EXEC sys.sp_addextendedproperty @name = N'Description',
  @value = N'Number of the day in the month', @level0type = N'SCHEMA',
  @level0name = N'dbo', @level1type = N'TABLE', @level1name = N'DT_Date',
  @level2type = N'COLUMN', @level2name = N'DayOfMonth' ;
EXEC sys.sp_addextendedproperty @name = N'Description',
  @value = N'Number of the day in the year', @level0type = N'SCHEMA',
  @level0name = N'dbo', @level1type = N'TABLE', @level1name = N'DT_Date',
  @level2type = N'COLUMN', @level2name = N'DayOfYear' ;
EXEC sys.sp_addextendedproperty @name = N'Description',
  @value = N'Is today a weekday or a weekend', @level0type = N'SCHEMA',
  @level0name = N'dbo', @level1type = N'TABLE', @level1name = N'DT_Date',
  @level2type = N'COLUMN', @level2name = N'WeekdayWeekend' ;
EXEC sys.sp_addextendedproperty @name = N'Description',
  @value = N'Week of year', @level0type = N'SCHEMA', @level0name = N'dbo',
  @level1type = N'TABLE', @level1name = N'DT_Date', @level2type = N'COLUMN',
  @level2name = N'WeekOfYear' ;
EXEC sys.sp_addextendedproperty @name = N'Description', @value = N'Month name',
  @level0type = N'SCHEMA', @level0name = N'dbo', @level1type = N'TABLE',
  @level1name = N'DT_Date', @level2type = N'COLUMN', @level2name = N'MonthName' ;
```

```
EXEC sys.sp_addextendedproperty @name = N'Description',
  @value = N'Month of year', @level0type = N'SCHEMA', @level0name = N'dbo',
  @level1type = N'TABLE', @level1name = N'DT_Date', @level2type = N'COLUMN',
  @level2name = N'MonthOfYear' ;
EXEC sys.sp_addextendedproperty @name = N'Description',
  @value = N'Is this the last day of the calendar month?',
  @level0type = N'SCHEMA', @level0name = N'dbo', @level1type = N'TABLE',
  @level1name = N'DT_Date', @level2type = N'COLUMN',
  @level2name = N'IsLastDayOfMonth' ;
EXEC sys.sp_addextendedproperty @name = N'Description',
  @value = N'Calendar quarter', @level0type = N'SCHEMA', @level0name = N'dbo',
  @level1type = N'TABLE', @level1name = N'DT_Date', @level2type = N'COLUMN',
  @level2name = N'CalendarQuarter' ;
EXEC sys.sp_addextendedproperty @name = N'Description',
  @value = N'Calendar year', @level0type = N'SCHEMA', @level0name = N'dbo',
  @level1type = N'TABLE', @level1name = N'DT_Date', @level2type = N'COLUMN',
  @level2name = N'CalendarYear' ;
EXEC sys.sp_addextendedproperty @name = N'Description',
  @value = N'Calendar year and month', @level0type = N'SCHEMA',
  @level0name = N'dbo', @level1type = N'TABLE', @level1name = N'DT_Date',
  @level2type = N'COLUMN', @level2name = N'CalendarYearMonth' ;
EXEC sys.sp_addextendedproperty @name = N'Description',
  @value = N'Calendar year and quarter', @level0type = N'SCHEMA',
  @level0name = N'dbo', @level1type = N'TABLE', @level1name = N'DT_Date',
  @level2type = N'COLUMN', @level2name = N'CalendarYearQtr' ;
EXEC sys.sp_addextendedproperty @name = N'Description',
  @value = N'Fiscal month of year (1..12). FY starts in July',
  @level0type = N'SCHEMA', @level0name = N'dbo', @level1type = N'TABLE',
  @level1name = N'DT_Date', @level2type = N'COLUMN',
  @level2name = N'FiscalMonthOfYear' ;
EXEC sys.sp_addextendedproperty @name = N'Description',
  @value = N'Fiscal quarter', @level0type = N'SCHEMA', @level0name = N'dbo',
  @level1type = N'TABLE', @level1name = N'DT_Date', @level2type = N'COLUMN',
  @level2name = N'FiscalQuarter' ;
EXEC sys.sp_addextendedproperty @name = N'Description',
  @value = N'Fiscal year. Fiscal year begins in July.',
  @level0type = N'SCHEMA', @level0name = N'dbo', @level1type = N'TABLE',
  @level1name = N'DT_Date', @level2type = N'COLUMN', @level2name = N'FiscalYear' ;
EXEC sys.sp_addextendedproperty @name = N'Description',
  @value = N'Fiscal year and month', @level0type = N'SCHEMA',
  @level0name = N'dbo', @level1type = N'TABLE', @level1name = N'DT_Date',
  @level2type = N'COLUMN', @level2name = N'FiscalYearMonth' ;
EXEC sys.sp_addextendedproperty @name = N'Description',
  @value = N'Fiscal year and quarter', @level0type = N'SCHEMA',
  @level0name = N'dbo', @level1type = N'TABLE', @level1name = N'DT_Date',
  @level2type = N'COLUMN', @level2name = N'FiscalYearQtr' ;
EXEC sys.sp_addextendedproperty @name = N'Description',
  @value = N'What process loaded this row?', @level0type = N'SCHEMA',
  @level0name = N'dbo', @level1type = N'TABLE', @level1name = N'DT_Date',
  @level2type = N'COLUMN', @level2name = N'AuditKey' ;
EXEC sys.sp_addextendedproperty @name = N'FK To',
  @value = N'DimAudit.AuditKey', @level0type = N'SCHEMA', @level0name = N'dbo',
  @level1type = N'TABLE', @level1name = N'DT_Date', @level2type = N'COLUMN',
  @level2name = N'AuditKey' ;
EXEC sys.sp_addextendedproperty @name = N'Example Values',
  @value = N'20041123', @level0type = N'SCHEMA', @level0name = N'dbo',
```

```
  @level1type = N'TABLE', @level1name = N'DT_Date', @level2type = N'COLUMN',
  @level2name = N'DateKey' ;
EXEC sys.sp_addextendedproperty @name = N'Example Values',
  @value = N'11/23/2004', @level0type = N'SCHEMA', @level0name = N'dbo',
  @level1type = N'TABLE', @level1name = N'DT_Date', @level2type = N'COLUMN',
  @level2name = N'FullDate' ;
EXEC sys.sp_addextendedproperty @name = N'Example Values',
  @value = N'23-Nov-2004', @level0type = N'SCHEMA', @level0name = N'dbo',
  @level1type = N'TABLE', @level1name = N'DT_Date', @level2type = N'COLUMN',
  @level2name = N'DateName' ;
EXEC sys.sp_addextendedproperty @name = N'Example Values', @value = N'1..7',
  @level0type = N'SCHEMA', @level0name = N'dbo', @level1type = N'TABLE',
  @level1name = N'DT_Date', @level2type = N'COLUMN', @level2name = N'DayOfWeek' ;
EXEC sys.sp_addextendedproperty @name = N'Example Values', @value = N'Sunday',
  @level0type = N'SCHEMA', @level0name = N'dbo', @level1type = N'TABLE',
  @level1name = N'DT_Date', @level2type = N'COLUMN',
  @level2name = N'DayNameOfWeek' ;
EXEC sys.sp_addextendedproperty @name = N'Example Values', @value = N'1..31',
  @level0type = N'SCHEMA', @level0name = N'dbo', @level1type = N'TABLE',
  @level1name = N'DT_Date', @level2type = N'COLUMN', @level2name = N'DayOfMonth' ;
EXEC sys.sp_addextendedproperty @name = N'Example Values', @value = N'1..365',
  @level0type = N'SCHEMA', @level0name = N'dbo', @level1type = N'TABLE',
  @level1name = N'DT_Date', @level2type = N'COLUMN', @level2name = N'DayOfYear' ;
EXEC sys.sp_addextendedproperty @name = N'Example Values',
  @value = N'Weekday, Weekend', @level0type = N'SCHEMA', @level0name = N'dbo',
  @level1type = N'TABLE', @level1name = N'DT_Date', @level2type = N'COLUMN',
  @level2name = N'WeekdayWeekend' ;
EXEC sys.sp_addextendedproperty @name = N'Example Values',
  @value = N'1..52 or 53', @level0type = N'SCHEMA', @level0name = N'dbo',
  @level1type = N'TABLE', @level1name = N'DT_Date', @level2type = N'COLUMN',
  @level2name = N'WeekOfYear' ;
EXEC sys.sp_addextendedproperty @name = N'Example Values',
  @value = N'November', @level0type = N'SCHEMA', @level0name = N'dbo',
  @level1type = N'TABLE', @level1name = N'DT_Date', @level2type = N'COLUMN',
  @level2name = N'MonthName' ;
EXEC sys.sp_addextendedproperty @name = N'Example Values',
  @value = N'1, 2, …, 12', @level0type = N'SCHEMA', @level0name = N'dbo',
  @level1type = N'TABLE', @level1name = N'DT_Date', @level2type = N'COLUMN',
  @level2name = N'MonthOfYear' ;
EXEC sys.sp_addextendedproperty @name = N'Example Values', @value = N'Y, N',
  @level0type = N'SCHEMA', @level0name = N'dbo', @level1type = N'TABLE',
  @level1name = N'DT_Date', @level2type = N'COLUMN',
  @level2name = N'IsLastDayOfMonth' ;
EXEC sys.sp_addextendedproperty @name = N'Example Values',
  @value = N'1, 2, 3, 4', @level0type = N'SCHEMA', @level0name = N'dbo',
  @level1type = N'TABLE', @level1name = N'DT_Date', @level2type = N'COLUMN',
  @level2name = N'CalendarQuarter' ;
EXEC sys.sp_addextendedproperty @name = N'Example Values', @value = N'2004',
  @level0type = N'SCHEMA', @level0name = N'dbo', @level1type = N'TABLE',
  @level1name = N'DT_Date', @level2type = N'COLUMN',
  @level2name = N'CalendarYear' ;
EXEC sys.sp_addextendedproperty @name = N'Example Values', @value = N'2004-01',
  @level0type = N'SCHEMA', @level0name = N'dbo', @level1type = N'TABLE',
  @level1name = N'DT_Date', @level2type = N'COLUMN',
  @level2name = N'CalendarYearMonth' ;
EXEC sys.sp_addextendedproperty @name = N'Example Values', @value = N'2004Q1',
```

```
  @level0type = N'SCHEMA', @level0name = N'dbo', @level1type = N'TABLE',
  @level1name = N'DT_Date', @level2type = N'COLUMN',
  @level2name = N'CalendarYearQtr' ;
EXEC sys.sp_addextendedproperty @name = N'Example Values',
  @value = N'1, 2, …, 12', @level0type = N'SCHEMA', @level0name = N'dbo',
  @level1type = N'TABLE', @level1name = N'DT_Date', @level2type = N'COLUMN',
  @level2name = N'FiscalMonthOfYear' ;
EXEC sys.sp_addextendedproperty @name = N'Example Values',
  @value = N'1, 2, 3, 4', @level0type = N'SCHEMA', @level0name = N'dbo',
  @level1type = N'TABLE', @level1name = N'DT_Date', @level2type = N'COLUMN',
  @level2name = N'FiscalQuarter' ;
EXEC sys.sp_addextendedproperty @name = N'Example Values', @value = N'2004',
  @level0type = N'SCHEMA', @level0name = N'dbo', @level1type = N'TABLE',
  @level1name = N'DT_Date', @level2type = N'COLUMN', @level2name = N'FiscalYear' ;
EXEC sys.sp_addextendedproperty @name = N'Example Values',
  @value = N'FY2004-01', @level0type = N'SCHEMA', @level0name = N'dbo',
  @level1type = N'TABLE', @level1name = N'DT_Date', @level2type = N'COLUMN',
  @level2name = N'FiscalYearMonth' ;
EXEC sys.sp_addextendedproperty @name = N'Example Values',
  @value = N'FY2004Q1', @level0type = N'SCHEMA', @level0name = N'dbo',
  @level1type = N'TABLE', @level1name = N'DT_Date', @level2type = N'COLUMN',
  @level2name = N'FiscalYearQtr' ;
EXEC sys.sp_addextendedproperty @name = N'SCD  Type', @value = N'1',
  @level0type = N'SCHEMA', @level0name = N'dbo', @level1type = N'TABLE',
  @level1name = N'DT_Date', @level2type = N'COLUMN', @level2name = N'DateName' ;
EXEC sys.sp_addextendedproperty @name = N'SCD  Type', @value = N'1',
  @level0type = N'SCHEMA', @level0name = N'dbo', @level1type = N'TABLE',
  @level1name = N'DT_Date', @level2type = N'COLUMN', @level2name = N'DayOfWeek' ;
EXEC sys.sp_addextendedproperty @name = N'SCD  Type', @value = N'1',
  @level0type = N'SCHEMA', @level0name = N'dbo', @level1type = N'TABLE',
  @level1name = N'DT_Date', @level2type = N'COLUMN',
  @level2name = N'DayNameOfWeek' ;
EXEC sys.sp_addextendedproperty @name = N'SCD  Type', @value = N'1',
  @level0type = N'SCHEMA', @level0name = N'dbo', @level1type = N'TABLE',
  @level1name = N'DT_Date', @level2type = N'COLUMN', @level2name = N'DayOfMonth' ;
EXEC sys.sp_addextendedproperty @name = N'SCD  Type', @value = N'1',
  @level0type = N'SCHEMA', @level0name = N'dbo', @level1type = N'TABLE',
  @level1name = N'DT_Date', @level2type = N'COLUMN', @level2name = N'DayOfYear' ;
EXEC sys.sp_addextendedproperty @name = N'SCD  Type', @value = N'1',
  @level0type = N'SCHEMA', @level0name = N'dbo', @level1type = N'TABLE',
  @level1name = N'DT_Date', @level2type = N'COLUMN',
  @level2name = N'WeekdayWeekend' ;
EXEC sys.sp_addextendedproperty @name = N'SCD  Type', @value = N'1',
  @level0type = N'SCHEMA', @level0name = N'dbo', @level1type = N'TABLE',
  @level1name = N'DT_Date', @level2type = N'COLUMN', @level2name = N'WeekOfYear' ;
EXEC sys.sp_addextendedproperty @name = N'SCD  Type', @value = N'1',
  @level0type = N'SCHEMA', @level0name = N'dbo', @level1type = N'TABLE',
  @level1name = N'DT_Date', @level2type = N'COLUMN', @level2name = N'MonthName' ;
EXEC sys.sp_addextendedproperty @name = N'SCD  Type', @value = N'1',
  @level0type = N'SCHEMA', @level0name = N'dbo', @level1type = N'TABLE',
  @level1name = N'DT_Date', @level2type = N'COLUMN', @level2name = N'MonthOfYear' ;
EXEC sys.sp_addextendedproperty @name = N'SCD  Type', @value = N'1',
  @level0type = N'SCHEMA', @level0name = N'dbo', @level1type = N'TABLE',
  @level1name = N'DT_Date', @level2type = N'COLUMN',
  @level2name = N'IsLastDayOfMonth' ;
EXEC sys.sp_addextendedproperty @name = N'SCD  Type', @value = N'1',
```

```
  @level0type = N'SCHEMA', @level0name = N'dbo', @level1type = N'TABLE',
  @level1name = N'DT_Date', @level2type = N'COLUMN',
  @level2name = N'CalendarQuarter' ;
EXEC sys.sp_addextendedproperty @name = N'SCD  Type', @value = N'1',
  @level0type = N'SCHEMA', @level0name = N'dbo', @level1type = N'TABLE',
  @level1name = N'DT_Date', @level2type = N'COLUMN',
  @level2name = N'CalendarYear' ;
EXEC sys.sp_addextendedproperty @name = N'SCD  Type', @value = N'1',
  @level0type = N'SCHEMA', @level0name = N'dbo', @level1type = N'TABLE',
  @level1name = N'DT_Date', @level2type = N'COLUMN',
  @level2name = N'CalendarYearMonth' ;
EXEC sys.sp_addextendedproperty @name = N'SCD  Type', @value = N'1',
  @level0type = N'SCHEMA', @level0name = N'dbo', @level1type = N'TABLE',
  @level1name = N'DT_Date', @level2type = N'COLUMN',
  @level2name = N'CalendarYearQtr' ;
EXEC sys.sp_addextendedproperty @name = N'SCD  Type', @value = N'1',
  @level0type = N'SCHEMA', @level0name = N'dbo', @level1type = N'TABLE',
  @level1name = N'DT_Date', @level2type = N'COLUMN',
  @level2name = N'FiscalMonthOfYear' ;
EXEC sys.sp_addextendedproperty @name = N'SCD  Type', @value = N'1',
  @level0type = N'SCHEMA', @level0name = N'dbo', @level1type = N'TABLE',
  @level1name = N'DT_Date', @level2type = N'COLUMN',
  @level2name = N'FiscalQuarter' ;
EXEC sys.sp_addextendedproperty @name = N'SCD  Type', @value = N'1',
  @level0type = N'SCHEMA', @level0name = N'dbo', @level1type = N'TABLE',
  @level1name = N'DT_Date', @level2type = N'COLUMN', @level2name = N'FiscalYear' ;
EXEC sys.sp_addextendedproperty @name = N'SCD  Type', @value = N'1',
  @level0type = N'SCHEMA', @level0name = N'dbo', @level1type = N'TABLE',
  @level1name = N'DT_Date', @level2type = N'COLUMN',
  @level2name = N'FiscalYearMonth' ;
EXEC sys.sp_addextendedproperty @name = N'SCD  Type', @value = N'1',
  @level0type = N'SCHEMA', @level0name = N'dbo', @level1type = N'TABLE',
  @level1name = N'DT_Date', @level2type = N'COLUMN',
  @level2name = N'FiscalYearQtr' ;
EXEC sys.sp_addextendedproperty @name = N'Source System', @value = N'Derived',
  @level0type = N'SCHEMA', @level0name = N'dbo', @level1type = N'TABLE',
  @level1name = N'DT_Date', @level2type = N'COLUMN', @level2name = N'DateKey' ;
EXEC sys.sp_addextendedproperty @name = N'Source System', @value = N'Derived',
  @level0type = N'SCHEMA', @level0name = N'dbo', @level1type = N'TABLE',
  @level1name = N'DT_Date', @level2type = N'COLUMN', @level2name = N'FullDate' ;
EXEC sys.sp_addextendedproperty @name = N'Source System', @value = N'Derived',
  @level0type = N'SCHEMA', @level0name = N'dbo', @level1type = N'TABLE',
  @level1name = N'DT_Date', @level2type = N'COLUMN', @level2name = N'DateName' ;
EXEC sys.sp_addextendedproperty @name = N'Source System', @value = N'Derived',
  @level0type = N'SCHEMA', @level0name = N'dbo', @level1type = N'TABLE',
  @level1name = N'DT_Date', @level2type = N'COLUMN', @level2name = N'DayOfWeek' ;
EXEC sys.sp_addextendedproperty @name = N'Source System', @value = N'Derived',
  @level0type = N'SCHEMA', @level0name = N'dbo', @level1type = N'TABLE',
  @level1name = N'DT_Date', @level2type = N'COLUMN',
  @level2name = N'DayNameOfWeek' ;
EXEC sys.sp_addextendedproperty @name = N'Source System', @value = N'Derived',
  @level0type = N'SCHEMA', @level0name = N'dbo', @level1type = N'TABLE',
  @level1name = N'DT_Date', @level2type = N'COLUMN', @level2name = N'DayOfMonth' ;
EXEC sys.sp_addextendedproperty @name = N'Source System', @value = N'Derived',
  @level0type = N'SCHEMA', @level0name = N'dbo', @level1type = N'TABLE',
  @level1name = N'DT_Date', @level2type = N'COLUMN', @level2name = N'DayOfYear' ;
```

```
EXEC sys.sp_addextendedproperty @name = N'Source System', @value = N'Derived',
  @level0type = N'SCHEMA', @level0name = N'dbo', @level1type = N'TABLE',
  @level1name = N'DT_Date', @level2type = N'COLUMN',
  @level2name = N'WeekdayWeekend' ;
EXEC sys.sp_addextendedproperty @name = N'Source System', @value = N'Derived',
  @level0type = N'SCHEMA', @level0name = N'dbo', @level1type = N'TABLE',
  @level1name = N'DT_Date', @level2type = N'COLUMN', @level2name = N'WeekOfYear' ;
EXEC sys.sp_addextendedproperty @name = N'Source System', @value = N'Derived',
  @level0type = N'SCHEMA', @level0name = N'dbo', @level1type = N'TABLE',
  @level1name = N'DT_Date', @level2type = N'COLUMN', @level2name = N'MonthName' ;
EXEC sys.sp_addextendedproperty @name = N'Source System', @value = N'Derived',
  @level0type = N'SCHEMA', @level0name = N'dbo', @level1type = N'TABLE',
  @level1name = N'DT_Date', @level2type = N'COLUMN', @level2name = N'MonthOfYear' ;
EXEC sys.sp_addextendedproperty @name = N'Source System', @value = N'Derived',
  @level0type = N'SCHEMA', @level0name = N'dbo', @level1type = N'TABLE',
  @level1name = N'DT_Date', @level2type = N'COLUMN',
  @level2name = N'IsLastDayOfMonth' ;
EXEC sys.sp_addextendedproperty @name = N'Source System', @value = N'Derived',
  @level0type = N'SCHEMA', @level0name = N'dbo', @level1type = N'TABLE',
  @level1name = N'DT_Date', @level2type = N'COLUMN',
  @level2name = N'CalendarQuarter' ;
EXEC sys.sp_addextendedproperty @name = N'Source System', @value = N'Derived',
  @level0type = N'SCHEMA', @level0name = N'dbo', @level1type = N'TABLE',
  @level1name = N'DT_Date', @level2type = N'COLUMN',
  @level2name = N'CalendarYear' ;
EXEC sys.sp_addextendedproperty @name = N'Source System', @value = N'Derived',
  @level0type = N'SCHEMA', @level0name = N'dbo', @level1type = N'TABLE',
  @level1name = N'DT_Date', @level2type = N'COLUMN',
  @level2name = N'CalendarYearMonth' ;
EXEC sys.sp_addextendedproperty @name = N'Source System', @value = N'Derived',
  @level0type = N'SCHEMA', @level0name = N'dbo', @level1type = N'TABLE',
  @level1name = N'DT_Date', @level2type = N'COLUMN',
  @level2name = N'CalendarYearQtr' ;
EXEC sys.sp_addextendedproperty @name = N'Source System', @value = N'Derived',
  @level0type = N'SCHEMA', @level0name = N'dbo', @level1type = N'TABLE',
  @level1name = N'DT_Date', @level2type = N'COLUMN',
  @level2name = N'FiscalMonthOfYear' ;
EXEC sys.sp_addextendedproperty @name = N'Source System', @value = N'Derived',
  @level0type = N'SCHEMA', @level0name = N'dbo', @level1type = N'TABLE',
  @level1name = N'DT_Date', @level2type = N'COLUMN',
  @level2name = N'FiscalQuarter' ;
EXEC sys.sp_addextendedproperty @name = N'Source System', @value = N'Derived',
  @level0type = N'SCHEMA', @level0name = N'dbo', @level1type = N'TABLE',
  @level1name = N'DT_Date', @level2type = N'COLUMN', @level2name = N'FiscalYear' ;
EXEC sys.sp_addextendedproperty @name = N'Source System', @value = N'Derived',
  @level0type = N'SCHEMA', @level0name = N'dbo', @level1type = N'TABLE',
  @level1name = N'DT_Date', @level2type = N'COLUMN',
  @level2name = N'FiscalYearMonth' ;
EXEC sys.sp_addextendedproperty @name = N'Source System', @value = N'Derived',
  @level0type = N'SCHEMA', @level0name = N'dbo', @level1type = N'TABLE',
  @level1name = N'DT_Date', @level2type = N'COLUMN',
  @level2name = N'FiscalYearQtr' ;
EXEC sys.sp_addextendedproperty @name = N'Source System',
  @value = N'Derived in ETL', @level0type = N'SCHEMA', @level0name = N'dbo',
  @level1type = N'TABLE', @level1name = N'DT_Date', @level2type = N'COLUMN',
  @level2name = N'AuditKey' ;
```

```
EXEC sys.sp_addextendedproperty @name = N'Comments',
  @value = N'In the form: yyyymmdd', @level0type = N'SCHEMA',
  @level0name = N'dbo', @level1type = N'TABLE', @level1name = N'DT_Date',
  @level2type = N'COLUMN', @level2name = N'DateKey' ;
GO



/*
          Now populating the kimball-based Date dimension table with data.
          The T-SQL approach below is an alternative to the Excel spreadsheet approach.
*/

USE [MartRental]
GO

SET NOCOUNT ON

-- Variables for holding user specified start and end dates
DECLARE @BeginDate DATETIME
DECLARE @EndDate DATETIME

-- Flag to determine if a date is the last day in the month
DECLARE @LastDayOfMon CHAR(1)

-- Offset to get to current fiscal date
DECLARE @FiscalYearMonthsOffset INT

-- Counters used in loop
DECLARE @DateCounter DATETIME    --Current date in loop
DECLARE @FiscalCounter DATETIME  --Fiscal Year Date in loop

-- Initialize the start and end date variables
SET @BeginDate = '01/01/2010'
SET @EndDate = '12/31/2020'


-- Using 6 for July of current year
SET @FiscalYearMonthsOffset = 6

-- Start the counter at the begin date
SET @DateCounter = @BeginDate

WHILE @DateCounter <= @EndDate
    BEGIN
        -- Calculate the current Fiscal date as an offset of
        -- the current date in the loop
        SET @FiscalCounter = DATEADD(m, @FiscalYearMonthsOffset, @DateCounter)

        -- Set value for IsLastDayOfMonth
        IF MONTH(@DateCounter) = MONTH(DATEADD(d, 1, @DateCounter))
          SET @LastDayOfMon = 'N'
        ELSE
          SET @LastDayOfMon = 'Y'

        -- add a record into the date dimension table for this date
```

```
        INSERT  INTO dbo.[DT_Date]
               (
                [DateKey]
               , [FullDate]
               , [DateName]
               , [DayOfWeek]
               , [DayNameOfWeek]
               , [DayOfMonth]
               , [DayOfYear]
               , [WeekdayWeekend]
               , [WeekOfYear]
               , [MonthName]
               , [MonthOfYear]
               , [IsLastDayOfMonth]
               , [CalendarQuarter]
               , [CalendarYear]
               , [CalendarYearMonth]
               , [CalendarYearQtr]
               , [FiscalMonthOfYear]
               , [FiscalQuarter]
               , [FiscalYear]
               , [FiscalYearMonth]
               , [FiscalYearQtr]
               )
        VALUES  (
                 ( YEAR(@DateCounter) * 10000 ) + ( MONTH(@DateCounter)
                                                    * 100 )
                 + DAY(@DateCounter)  --DateKey
               , @DateCounter -- FullDate
               , CAST(YEAR(@DateCounter) AS CHAR(4)) + '/'
                 + RIGHT('00' + RTRIM(CAST(DATEPART(mm, @DateCounter) AS CHAR(2))), 2) + '/'
                 + RIGHT('00' + RTRIM(CAST(DATEPART(dd, @DateCounter) AS CHAR(2))), 2) --DateName
               , DATEPART(dw, @DateCounter) --DayOfWeek
               , DATENAME(dw, @DateCounter) --DayNameOfWeek
               , DATENAME(dd, @DateCounter) --DayOfMonth
               , DATENAME(dy, @DateCounter) --DayOfYear
               , CASE DATENAME(dw, @DateCounter)
                   WHEN 'Saturday' THEN 'Weekend'
                   WHEN 'Sunday' THEN 'Weekend'
                   ELSE 'Weekday'
                 END --WeekdayWeekend
               , DATENAME(ww, @DateCounter) --WeekOfYear
               , DATENAME(mm, @DateCounter) --MonthName
               , MONTH(@DateCounter) --MonthOfYear
               , @LastDayOfMon --IsLastDayOfMonth
               , DATENAME(qq, @DateCounter) --CalendarQuarter
               , YEAR(@DateCounter) --CalendarYear
               , CAST(YEAR(@DateCounter) AS CHAR(4)) + '-'
                 + RIGHT('00' + RTRIM(CAST(DATEPART(mm, @DateCounter) AS CHAR(2))), 2) --
CalendarYearMonth
               , CAST(YEAR(@DateCounter) AS CHAR(4)) + 'Q' + DATENAME(qq, @DateCounter) --
CalendarYearQtr
               , MONTH(@FiscalCounter) --[FiscalMonthOfYear]
               , DATENAME(qq, @FiscalCounter) --[FiscalQuarter]
               , YEAR(@FiscalCounter) --[FiscalYear]
               , CAST(YEAR(@FiscalCounter) AS CHAR(4)) + '-'
```

```
                + RIGHT('00' + RTRIM(CAST(DATEPART(mm, @FiscalCounter) AS CHAR(2))), 2) --
[FiscalYearMonth]
                , CAST(YEAR(@FiscalCounter) AS CHAR(4)) + 'Q' + DATENAME(qq, @FiscalCounter) --
[FiscalYearQtr]
                )

        -- Increment the date counter for next pass thru the loop
        SET @DateCounter = DATEADD(d, 1, @DateCounter)
    END

-- Show all records for the final year specified for debugging purposes
SELECT  *
FROM [dbo].[DT_Date]
WHERE DateKey > (YEAR(@EndDate) * 10000)
GO


-- Add record for general invalid/unknown date
INSERT  INTO [dbo].[DT_Date]
      ( DateKey
      , FullDate
      , [DateName]
      , [DayOfWeek]
      , DayNameOfWeek
      , [DayOfMonth]
      , [DayOfYear]
      , WeekdayWeekend
      , WeekOfYear
      , [MonthName]
      , MonthOfYear
      , IsLastDayOfMonth
      , CalendarQuarter
      , CalendarYear
      , CalendarYearMonth
      , CalendarYearQtr
      , FiscalMonthOfYear
      , FiscalQuarter
      , FiscalYear
      , FiscalYearMonth
      , FiscalYearQtr
      )
VALUES  ( -1
      , NULL
      , 'Unknown'
      , NULL
      , 'Unknown'
      , NULL
      , NULL
      , 'Unknown'
      , NULL
      , 'Unknown'
      , NULL
      , 'N'
      , NULL
      , NULL
      , 'Unknown'
```

```
        , 'Unknown'
        , NULL
        , NULL
        , NULL
        , 'Unknown'
        , 'Unknown'
        )
GO

SET NOCOUNT OFF
GO
```

APPENDIX B

SCRIPT TASK FOR GENERATING HASH KEY FOR DV 2.0

MD5 Hash Script Task (in C#) for DV 2.0
(Hash function shown for Agent Business Concept – similar for other business concepts)

```
#region Namespaces
using System;
using System.Data;
using System.Security.Cryptography;
using System.Text;
using System.Text.RegularExpressions;
using Microsoft.SqlServer.Dts.Pipeline.Wrapper;
using Microsoft.SqlServer.Dts.Runtime.Wrapper;
#endregion

/// <summary>
/// This is the class to which to add your code.  Do not change the name, attributes, or parent
/// of this class.
/// </summary>
[Microsoft.SqlServer.Dts.Pipeline.SSISScriptComponentEntryPointAttribute]
public class ScriptMain : UserComponent
{ /// <summary>
   /// Generates a Hash Key using the Business Key as input
   /// <param name="businessKey"></param>
   /// <returns> A Hash for the Business Key</returns>
   /// </summary>

   private string GenerateHashKey(string businessKey)
   {
      string hashValue = "";

      //SHA256 crypObj = new SHA256CryptoServiceProvider();
      MD5 crypObj = new MD5CryptoServiceProvider();

      hashValue = BitConverter.ToString(crypObj.ComputeHash(ASCIIEncoding.ASCII.GetBytes(businessKey)));

      // Finally remove the hyphens from the hash
      hashValue = Regex.Replace(hashValue, "-", "").ToUpper();

      return hashValue;
   }


   /// <summary>
   /// This method is called once for every row that passes through the component from Input0.
   ///
   /// </summary>
   /// <param name="Row">The row that is currently passing through the component</param>
   public override void Input0_ProcessInputRow(Input0Buffer Row)
   {
      // Assign the value for the Hash Key
      Row.AgentHsk = GenerateHashKey(Row.AgentId.ToString());
```

```
    }
}
```

APPENDIX C

HANDLING PERMISSIBLE LOAD ERRORS IN DV TO DM LOADS

(Only DV 2.0 shown. DV 1.0 is similarly done by using sequence keys instead of hash keys)

DV 2.0 to DM Load for Customer Dimension

```
SELECT h.Customer_Bk, s.FirstName + ' '+  s.Lastname AS Name,
          s.DateOfBirth, s.City, s.ZipCode, s.State
FROM  Hub_Customer h, Sat_Customer s
WHERE s.Valid_To_Date IS NULL
AND  h.Customer_Hsk = s.Customer_Hsk

-- Business Rule redarding Customer
AND   h.Customer_Hsk NOT IN (SELECT Customer_Hsk
                             FROM Sat_Customer
                             WHERE Valid_To_Date IS NULL
                             AND DATEDIFF("yy", DateOfBirth, GETDATE()) < 18)
```

DV 2.0 to DM Load for Vehicle Dimension

```
SELECT h.Vehicle_Bk, s.Make,
           s.Model, s.Year, s.[Color]
FROM  Hub_Vehicle h, Sat_Vehicle s
WHERE s.Valid_To_Date IS NULL
AND h.Vehicle_Hsk = s.Vehicle_Hsk

-- Business Rule redarding Vehicle
AND   h.Vehicle_Hsk  NOT IN (SELECT Vehicle_Hsk
                             FROM Sat_Vehicle
                             WHERE Valid_To_Date IS NULL
                             AND (YEAR(GETDATE()) - Year > 3) OR (Year - YEAR(GETDATE()) > 1))
```

DV 2.0 to DM Load for Rental Fact

```
SELECT  YEAR(sR.RentalDate)*10000 + MONTH(sR.RentalDate)*100 + DAY(sR.RentalDate) AS
RentalDate_ID,
           hA.Agent_Bk, hC.Customer_Bk, hV.Vehicle_Bk, sR.NumDays, sR.Cost
FROM Link_Rental lR, Sat_Rental sR, Hub_Agent hA, Hub_Customer hC, Hub_Vehicle hV
WHERE lR.Rental_Hsk = sR.Rental_Hsk AND (sR.Valid_To_Date IS NULL)
AND   lR.Agent_Hsk = hA.Agent_Hsk
AND   lR.Customer_Hsk = hC.Customer_Hsk
AND   lR.Vehicle_Hsk = hV.Vehicle_Hsk
```

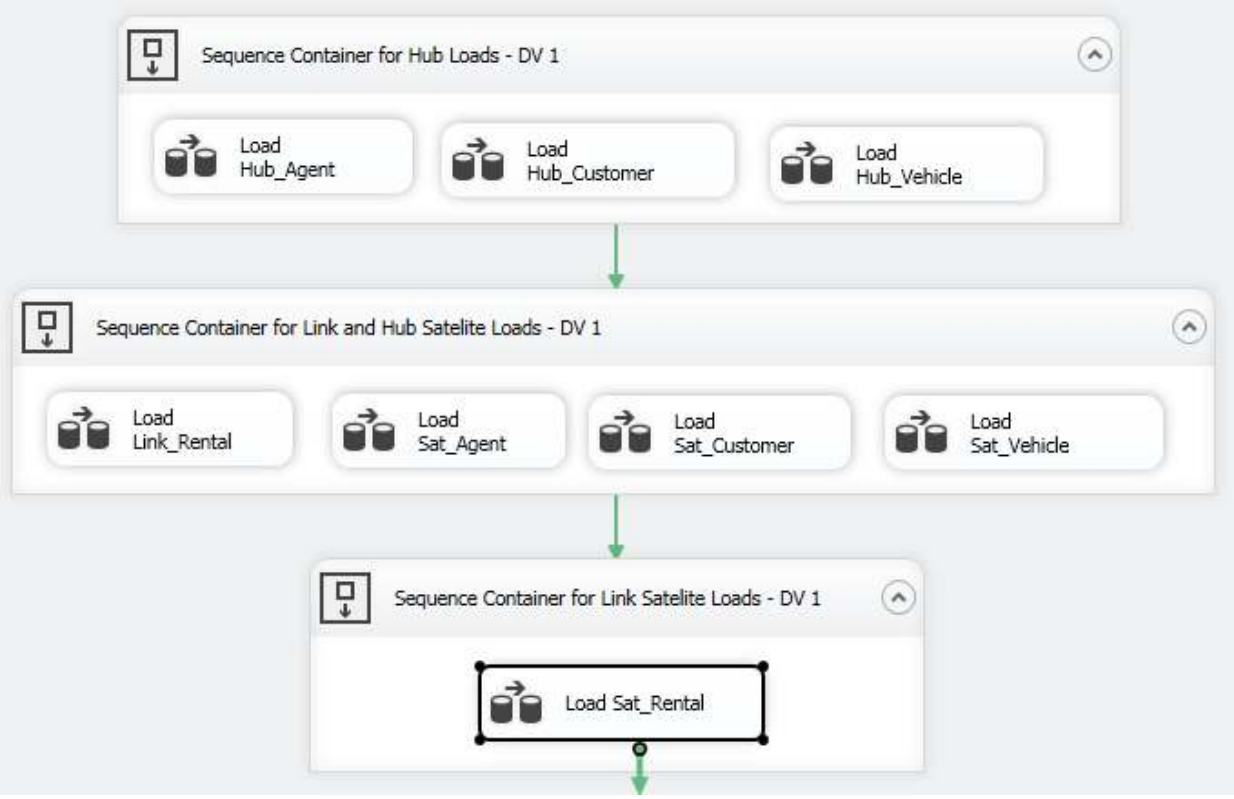
```
-- Business Rule redarding Customer
AND   hC.Customer_Hsk NOT IN (SELECT Customer_Hsk
                              FROM Sat_Customer
                              WHERE Valid_To_Date IS NULL
                              AND DATEDIFF("yy", DateOfBirth, GETDATE()) < 18)

-- Business Rule redarding Vehicle
AND   hV.Vehicle_Hsk  NOT IN (SELECT Vehicle_Hsk
                               FROM Sat_Vehicle
                               WHERE Valid_To_Date IS NULL
                               AND (YEAR(GETDATE()) - Year > 3) OR (Year - YEAR(GETDATE()) > 1))

 -- Business Rule regarding Rental
AND  (sR.NumDays >= 0 AND SR.NumDays <= 30)
```

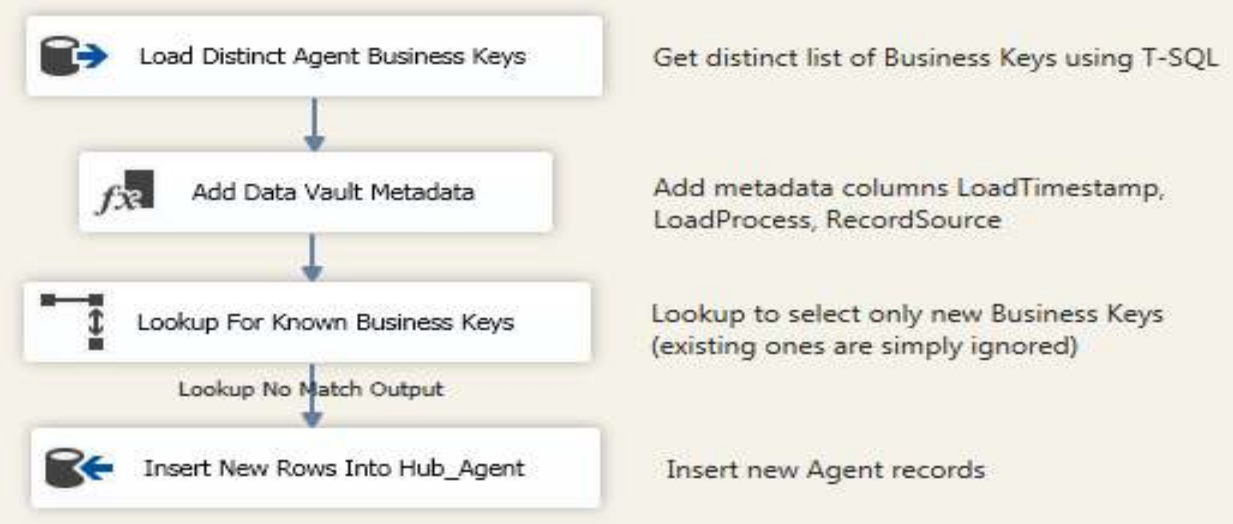
APPENDIX D

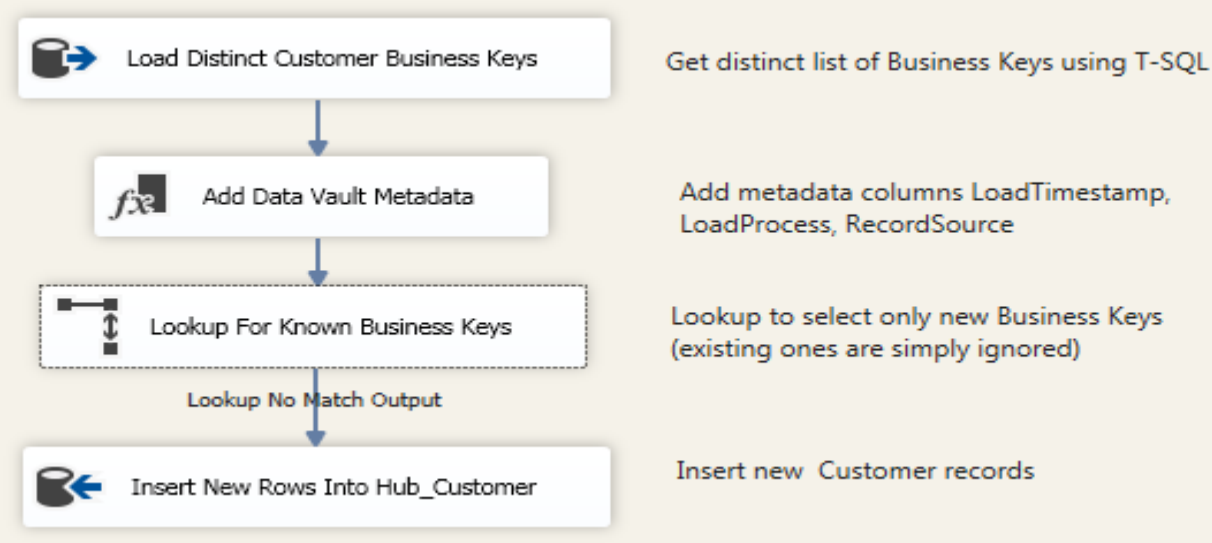
ETL WORK FLOWS

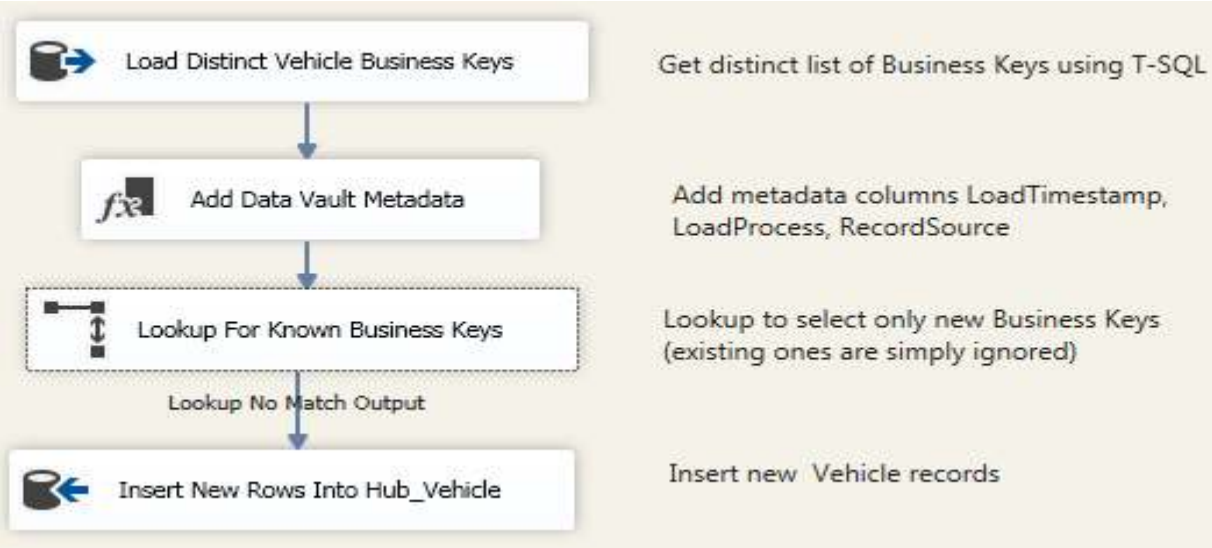
(The work flows for the four SSIS ETL packages that handle Permissible Load Errors shown below. The workflows for Rejecting all Load Errors not shown as they are similar - difference lies mainly in ETL logic which rejects all load errors)

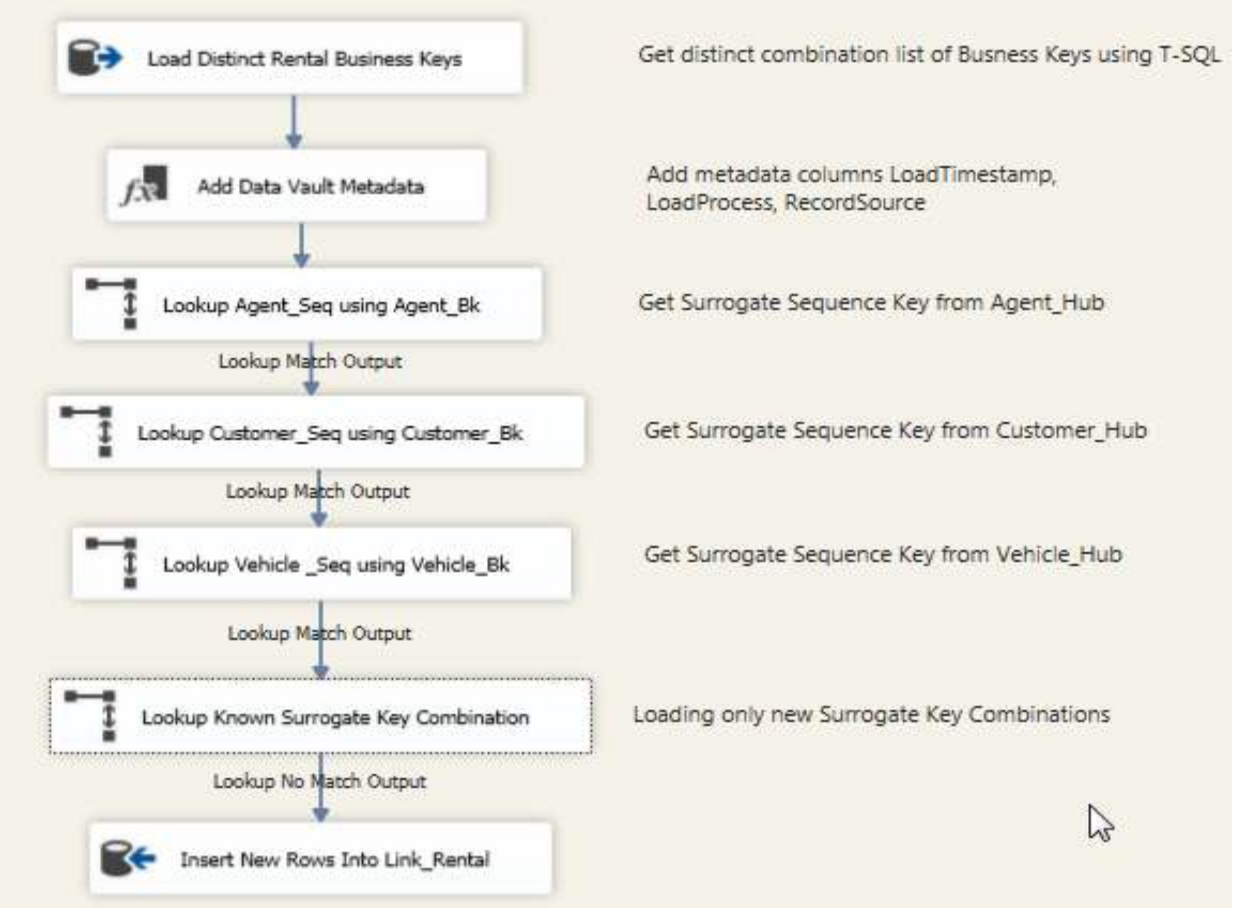
DV1_EDW_Load.dtsx

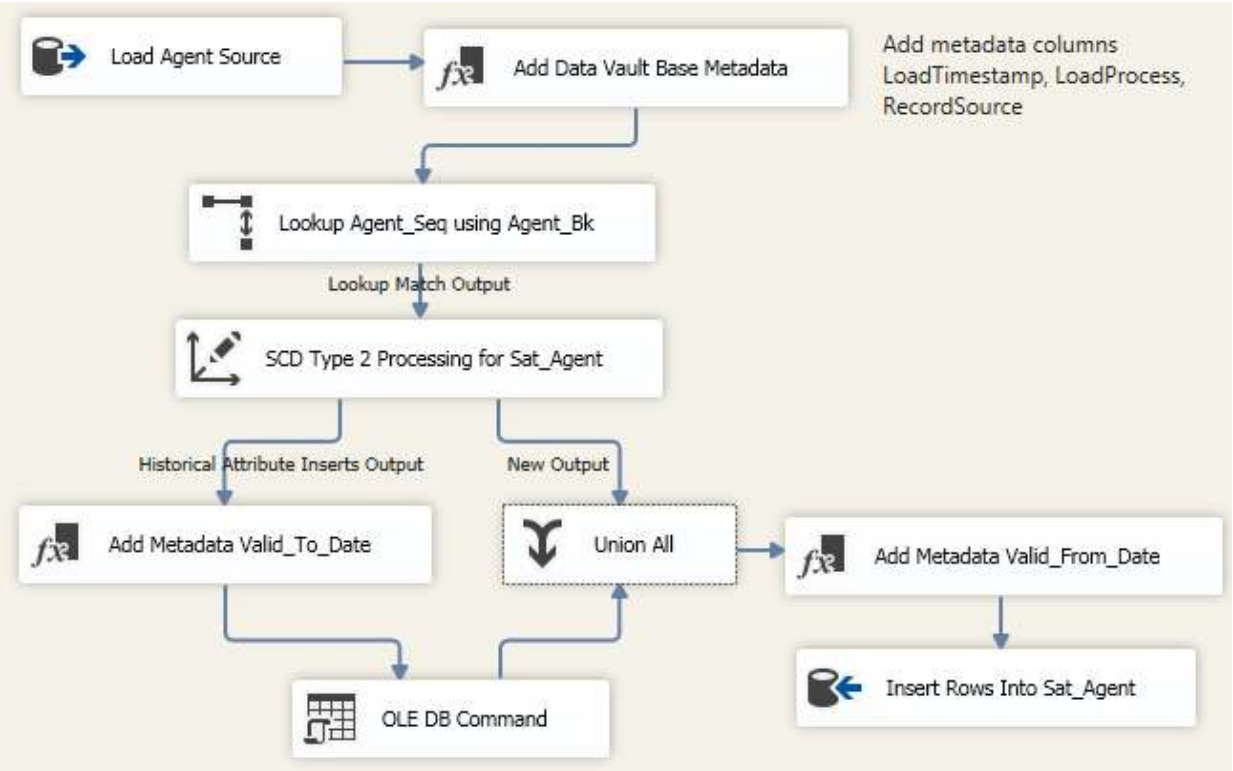


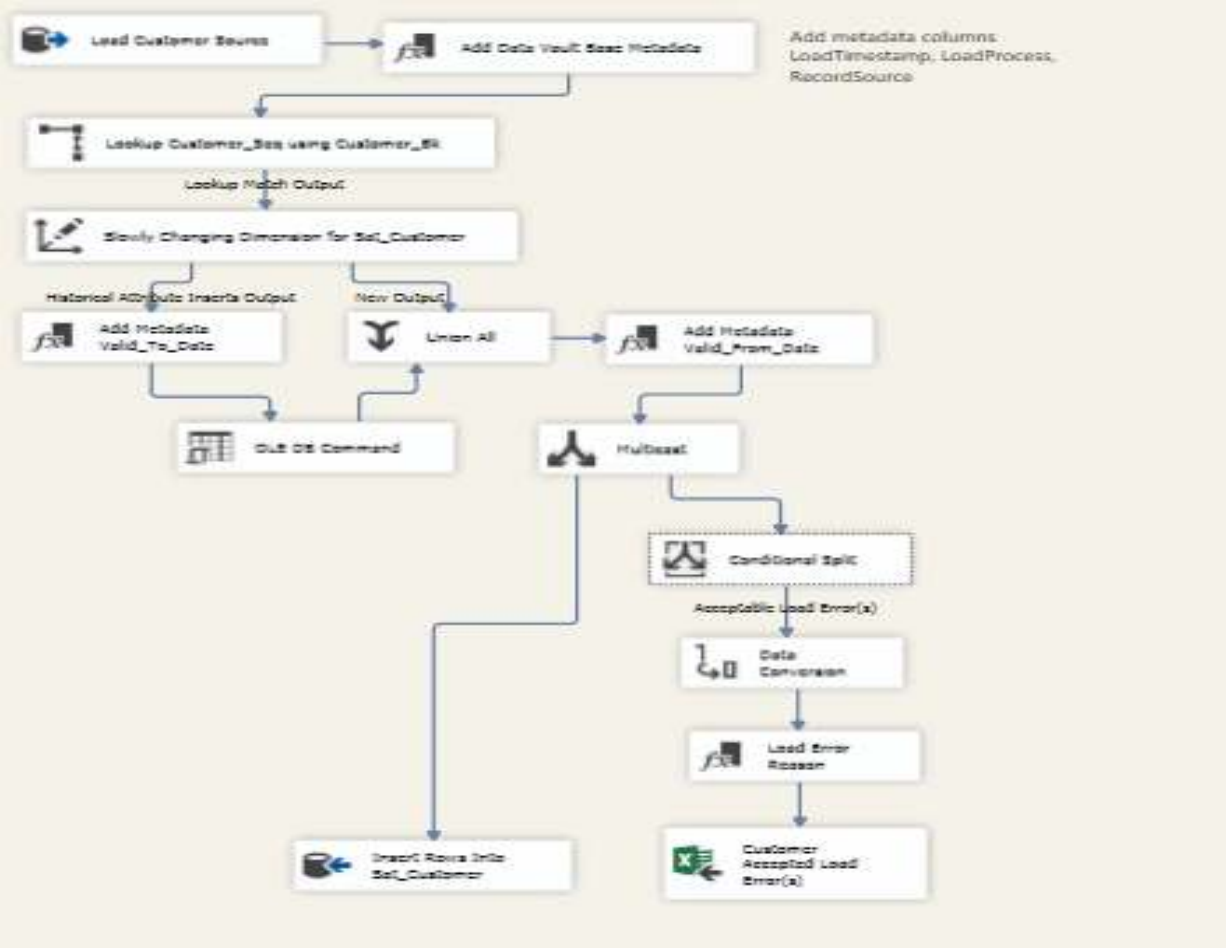

Load Distinct Agent Business Keys
Get distinct list of Business Keys using T-SQL
Add Data Vault Metadata
Add metadata columns LoadTimestamp, LoadProcess, RecordSource
Lookup For Known Business Keys
Lookup to select only new Business Keys (existing ones are simply ignored)
Lookup No Match Output
Insert New Rows Into Hub_Agent
Insert new Agent records

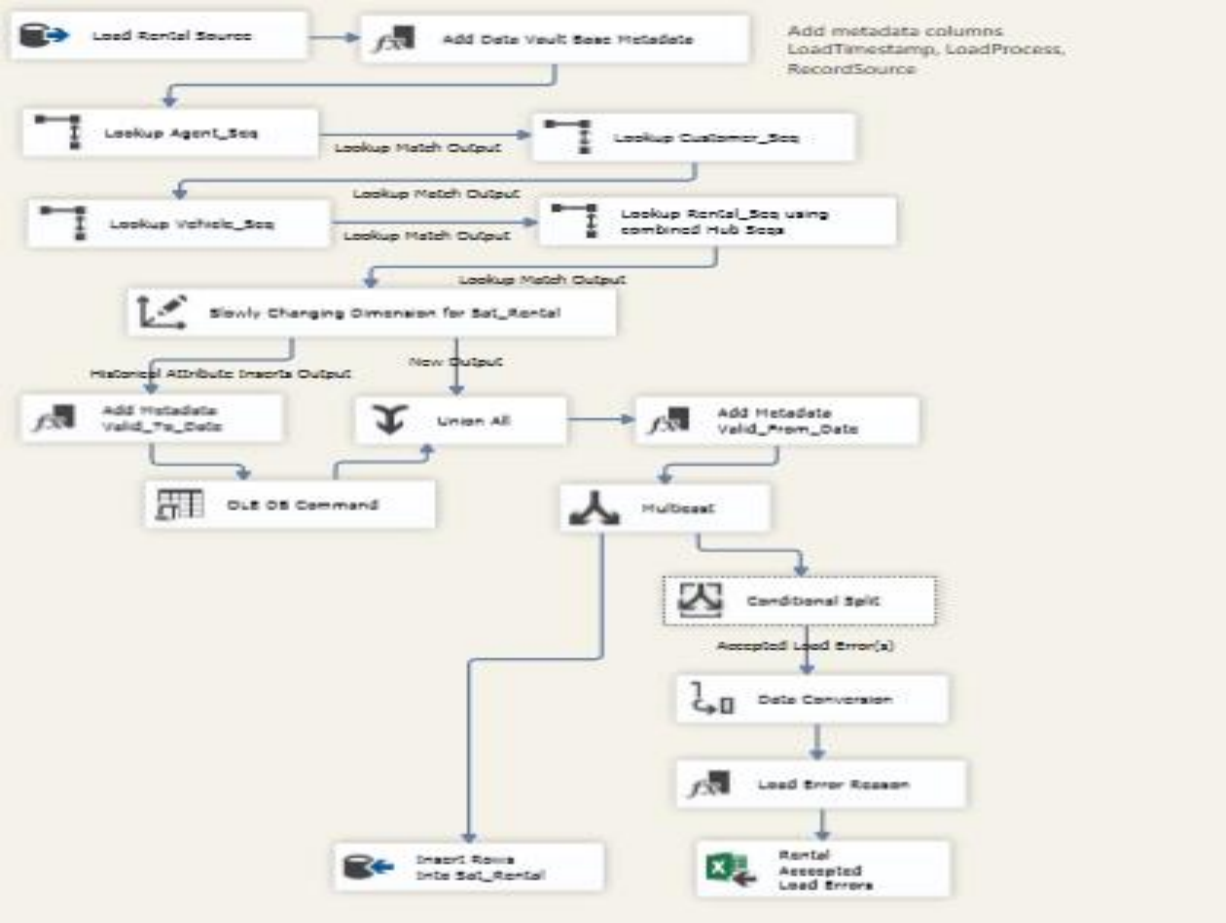

Load Distinct Customer Business Keys
Get distinct list of Business Keys using T-SQL
Add Data Vault Metadata
Add metadata columns LoadTimestamp, LoadProcess, RecordSource
Lookup For Known Business Keys
Lookup to select only new Business Keys (existing ones are simply ignored)
Lookup No Match Output
Insert New Rows Into Hub_Customer
Insert new Customer records

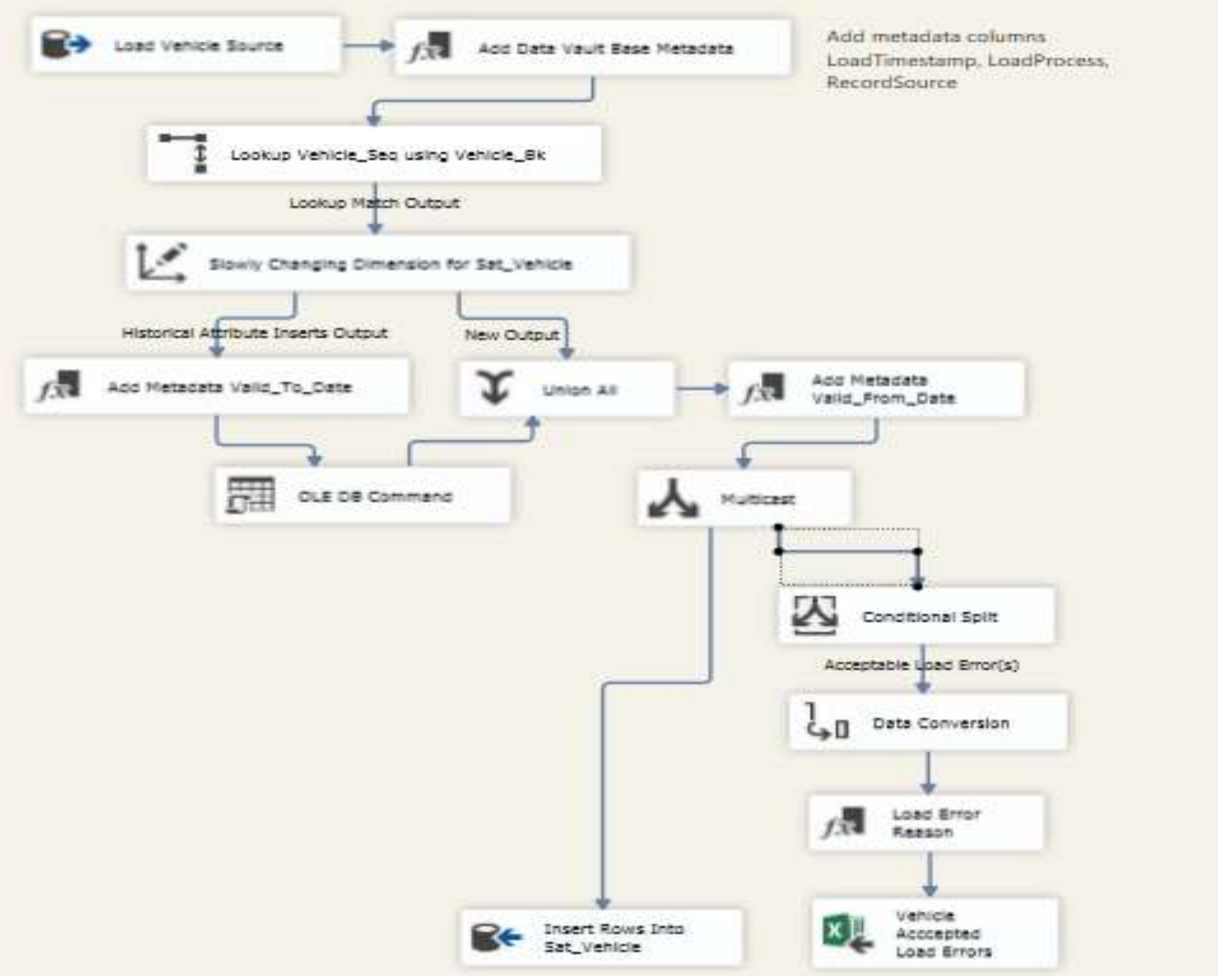

Load Distinct Vehicle Business Keys
Get distinct list of Business Keys using T-SQL
Add Data Vault Metadata
Add metadata columns LoadTimestamp, LoadProcess, RecordSource
Lookup For Known Business Keys
Lookup to select only new Business Keys (existing ones are simply ignored)
Lookup No Match Output
Insert New Rows Into Hub_Vehicle
Insert new Vehicle records


Load Distinct Rental Business Keys
Get distinct combination list of Busness Keys using T-SQL
Add Data Vault Metadata
Add metadata columns LoadTimestamp, LoadProcess, RecordSource
Lookup Agent_Seq using Agent_Bk
Get Surrogate Sequence Key from Agent_Hub
Lookup Match Output
Lookup Customer_Seq using Customer_Bk
Get Surrogate Sequence Key from Customer_Hub
Lookup Match Output
Lookup Vehicle _Seq using Vehicle_Bk
Get Surrogate Sequence Key from Vehicle_Hub
Lookup Match Output
Lookup Known Surrogate Key Combination
Loading only new Surrogate Key Combinations
Lookup No Match Output
Insert New Rows Into Link_Rental

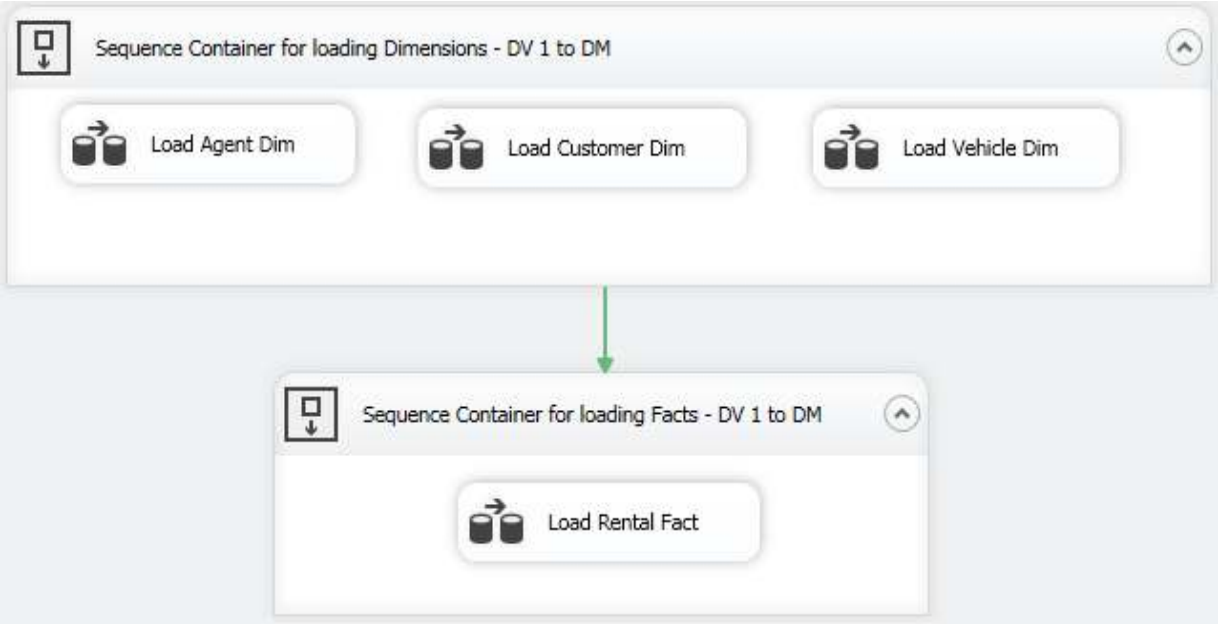

Load Agent Source
Add Data Vault Base Metadata
Add metadata columns LoadTimestamp, LoadProcess, RecordSource
Lookup Agent_Seq using Agent_Bk
Lookup Match Output
SCD Type 2 Processing for Sat_Agent
Historical Attribute Inserts Output
New Output
Add Metadata Valid_To_Date
Union All
Add Metadata Valid_From_Date
Insert Rows Into Sat_Agent
OLE DB Command

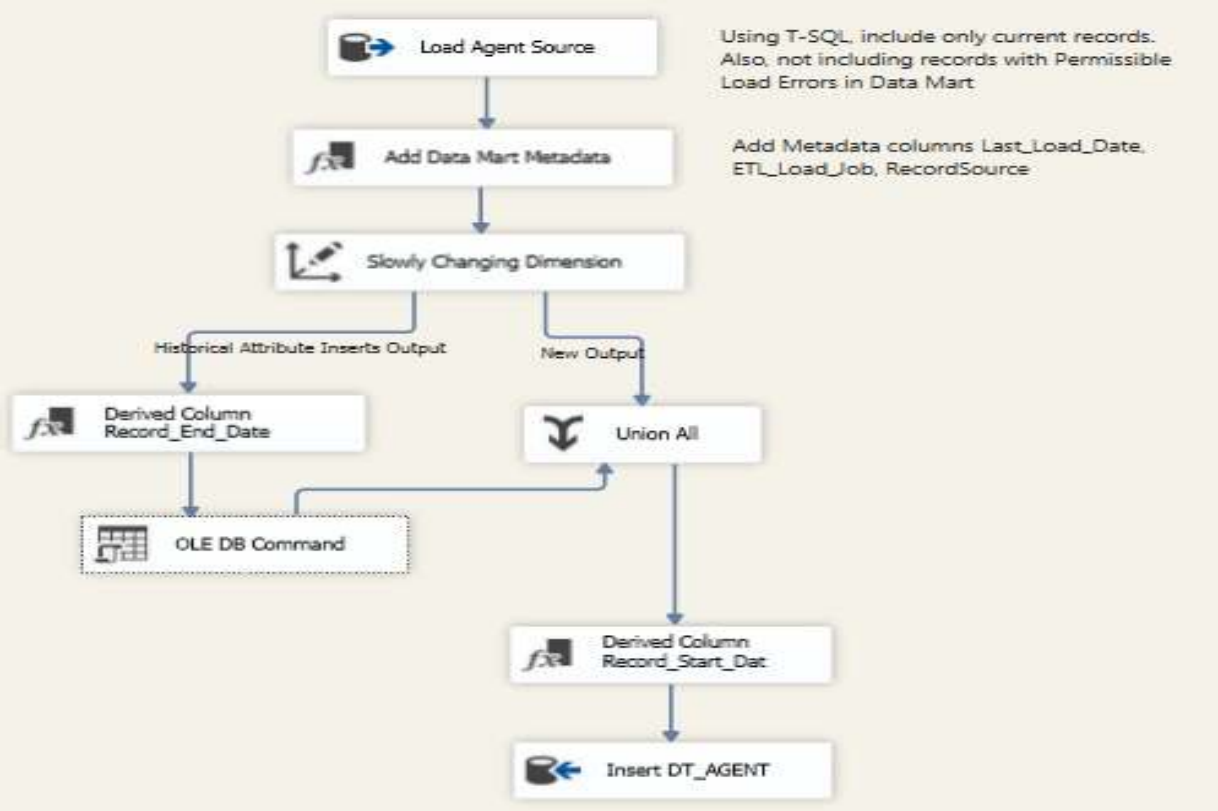

Load Customer Source
Add Data Vault Base Metadata
Add metadata columns
LoadTimestamp, LoadProcess,
RecordSource
Lookup Customer_Bsq using Customer_BK
Lookup Match Output
Slowly Changing Dimension for Sat_Customer
Historical Attribute Inserts Output
New Output
Add Metadata
Valid_To_Date
Union All
Add Metadata
Valid_From_Date
OLE DB Command
Multicast
Conditional Split
Acceptable Load Error(s)
Data
Conversion
Load Error
Reason
Insert Rows into
Sat_Customer
Customer
Accepted Load
Error(s)

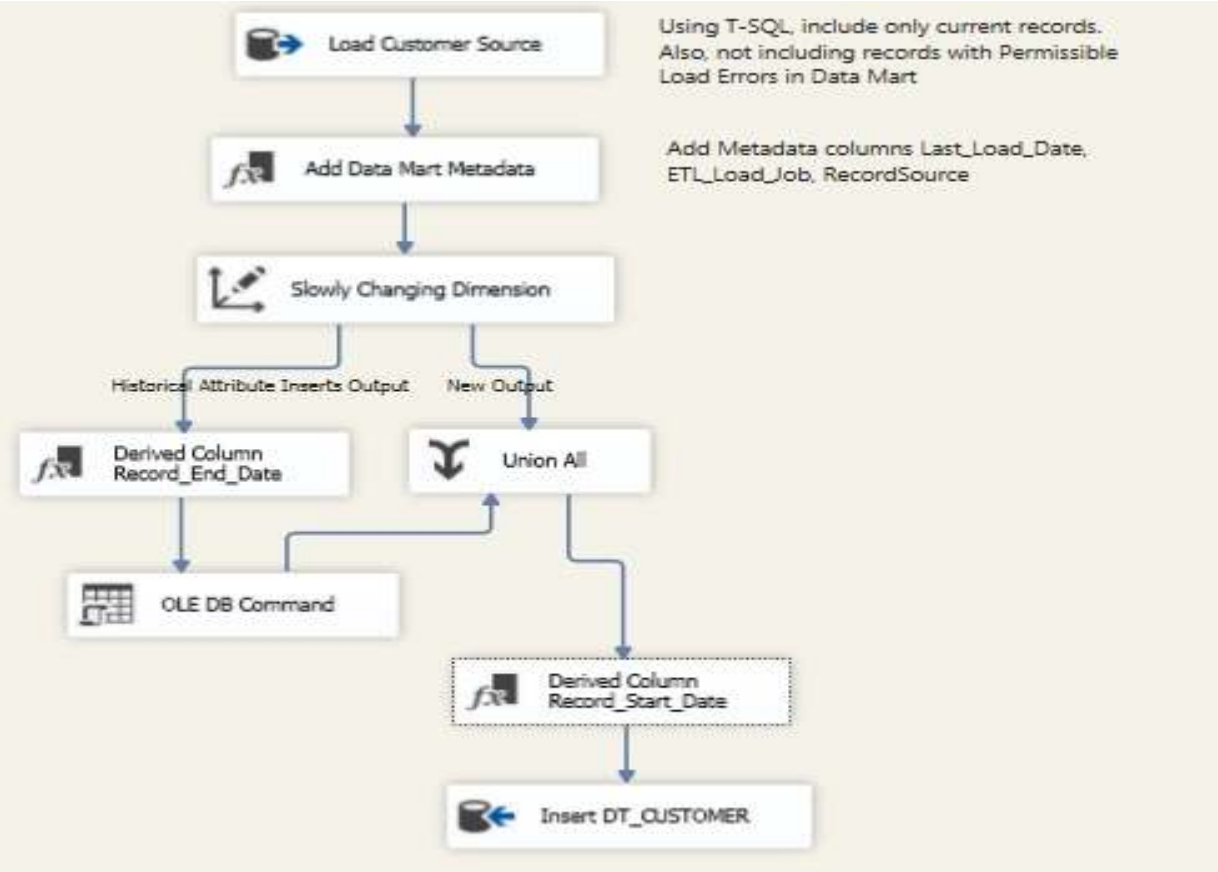

Load Rental Source
Add Data Vault Base Metadata
Add metadata columns
LoadTimestamp, LoadProcess,
RecordSource
Lookup Agent_Sca
Lookup Match Output
Lookup Customer_Sca
Lookup Match Output
Lookup Vehicle_Sca
Lookup Match Output
Lookup Rental_Sca using
combined Hub Scas
Lookup Match Output
Slowly Changing Dimension for Sat_Rental
Historical Attribute Inserts Output
New Output
Add Metadata
Valid_To_Date
Union All
Add Metadata
Valid_From_Date
OLE DB Command
Multicast
Conditional Split
Accepted Load Error(s)
Data Conversion
Load Error Reason
Insert Rows
into Sat_Rental
Rental
Accepted
Load Errors

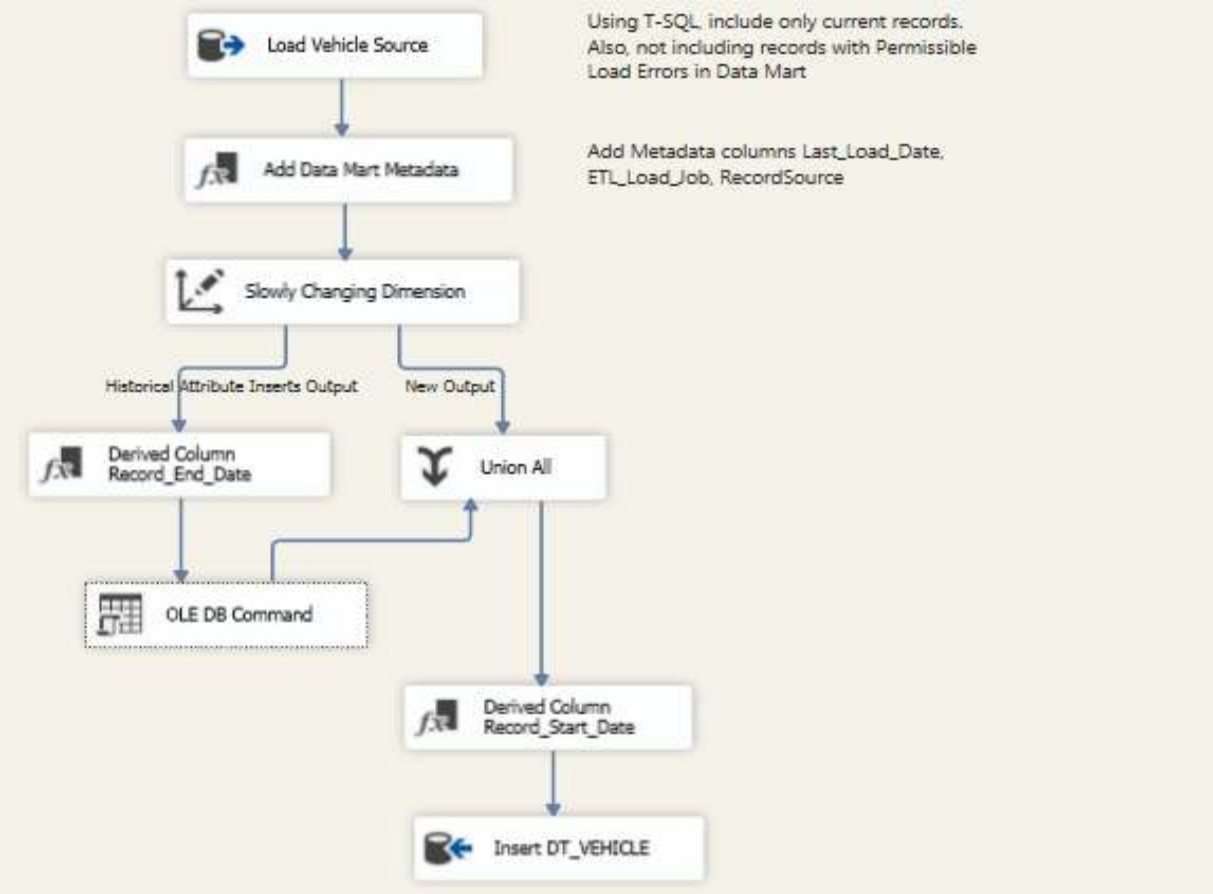


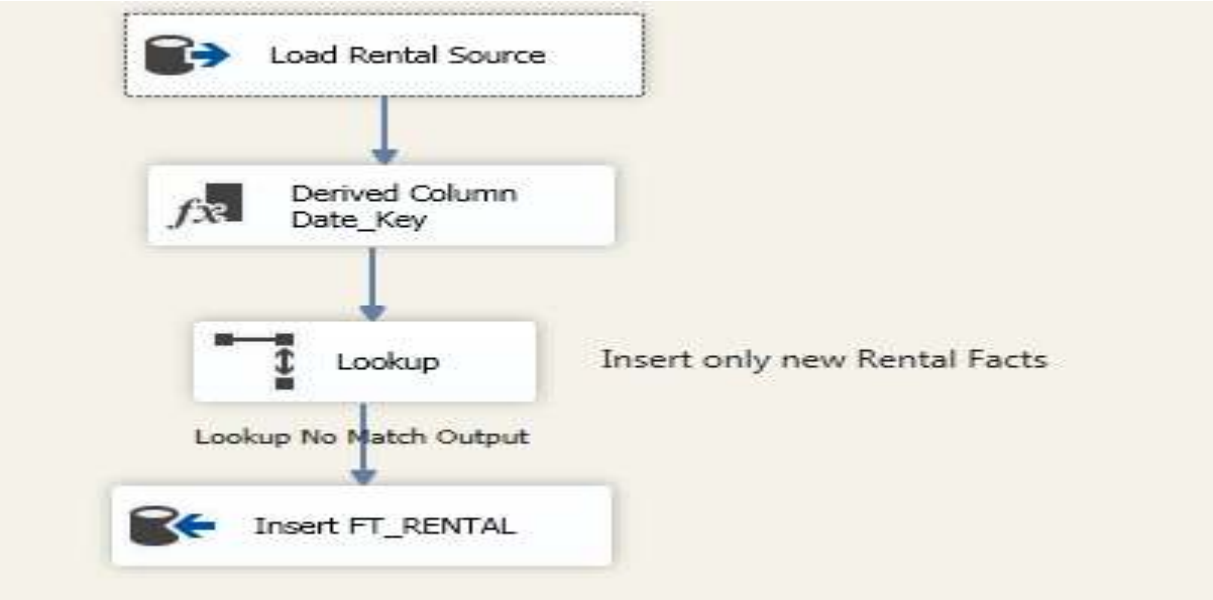
DV1_DM_Load.dtsx



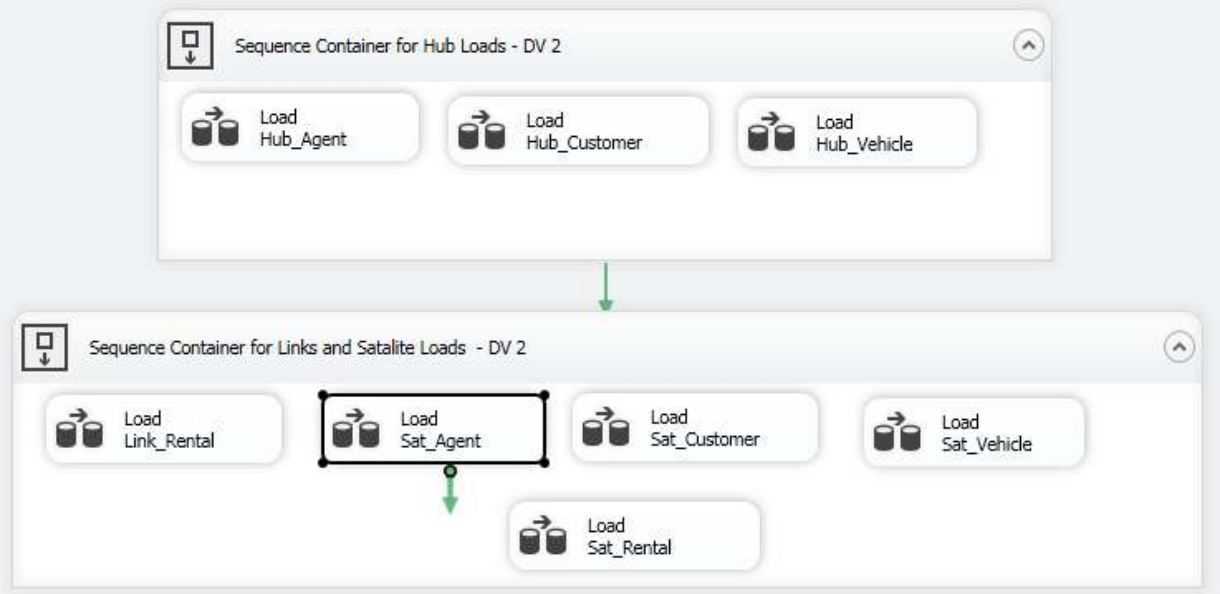

Load Agent Source
Using T-SQL, include only current records. Also, not including records with Permissible Load Errors in Data Mart
Add Data Mart Metadata
Add Metadata columns Last_Load_Date, ETL_Load_Job, RecordSource
Slowly Changing Dimension
Historical Attribute Inserts Output
New Output
Derived Column Record_End_Date
Union All
OLE DB Command
Derived Column Record_Start_Dat
Insert DT_AGENT

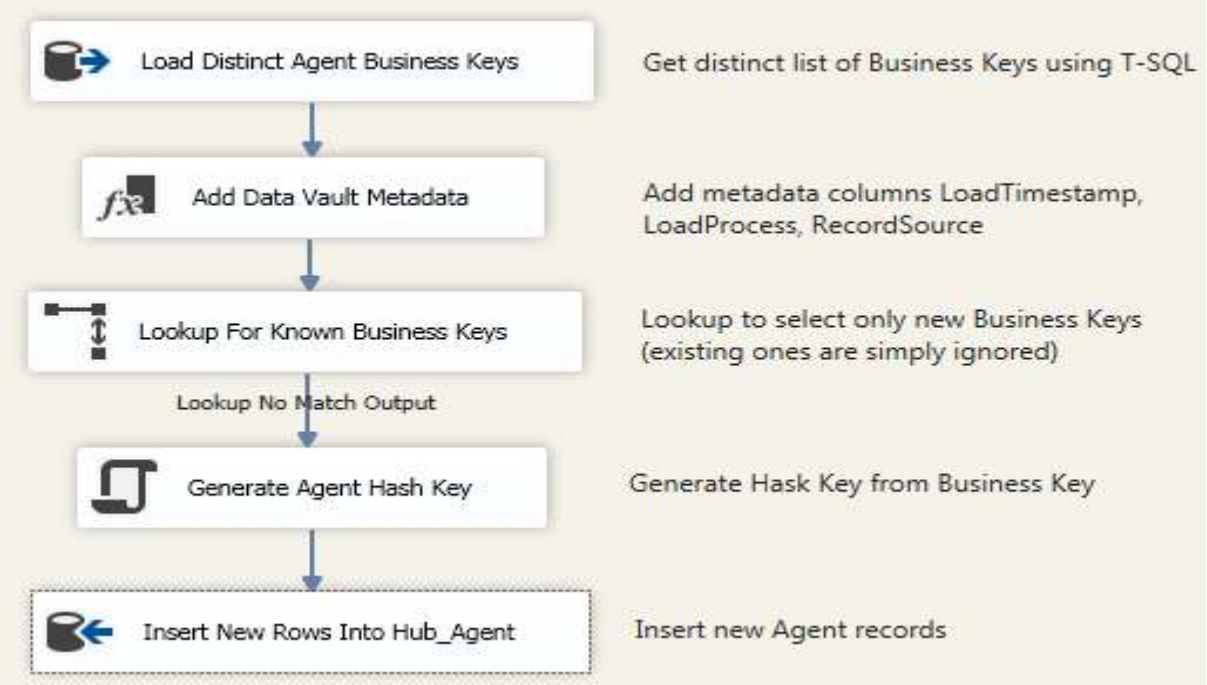

Load Customer Source
Using T-SQL, include only current records. Also, not including records with Permissible Load Errors in Data Mart
Add Data Mart Metadata
Add Metadata columns Last_Load_Date, ETL_Load_Job, RecordSource
Slowly Changing Dimension
Historical Attribute Inserts Output
New Output
Derived Column Record_End_Date
Union All
OLE DB Command
Derived Column Record_Start_Date
Insert DT_CUSTOMER

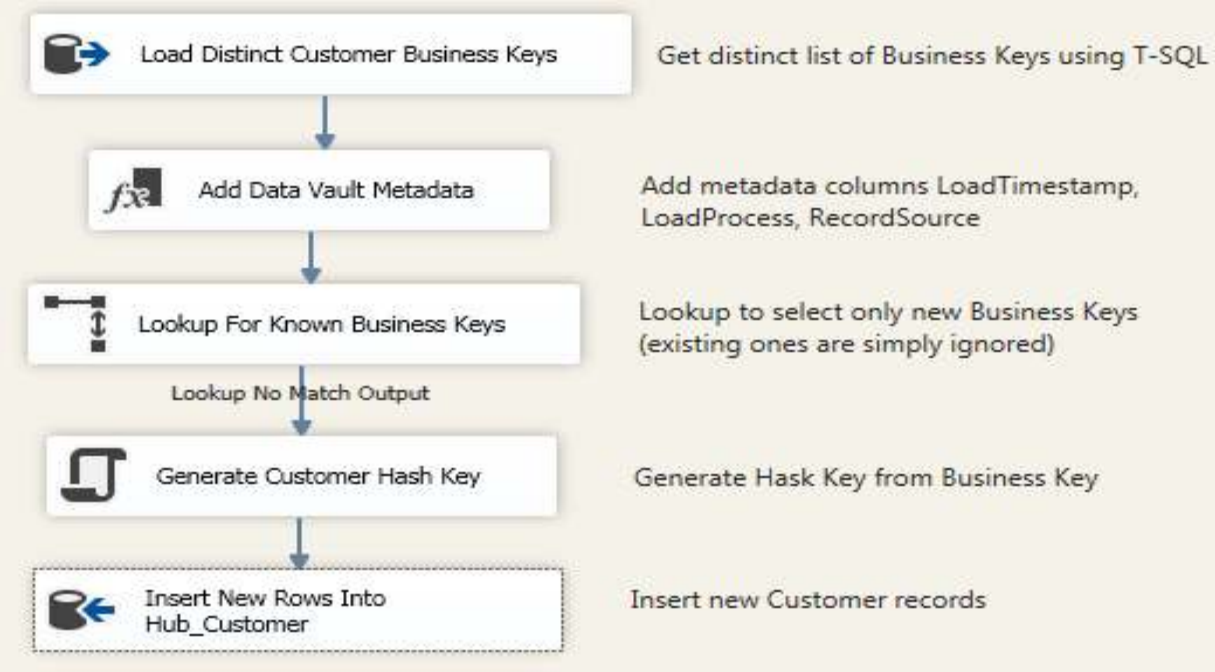

Load Vehicle Source
Using T-SQL include only current records.
Also, not including records with Permissible
Load Errors in Data Mart
Add Data Mart Metadata
Add Metadata columns Last_Load_Date,
ETL_Load_Job, RecordSource
Slowly Changing Dimension
Historical Attribute Inserts Output
New Output
Derived Column
Record_End_Date
Union All
OLE DB Command
Derived Column
Record_Start_Date
Insert DT_VEHICLE

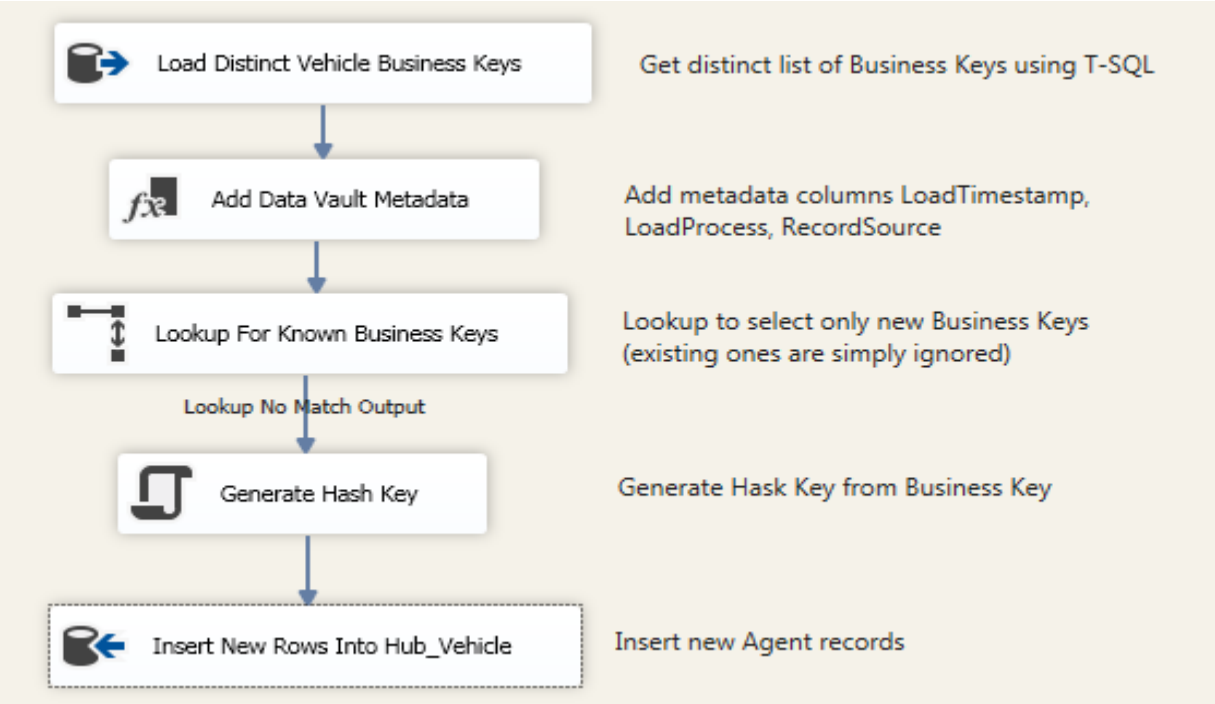

Load Rental Source
Derived Column
Date_Key
Lookup
Insert only new Rental Facts
Lookup No Match Output
Insert FT_RENTAL

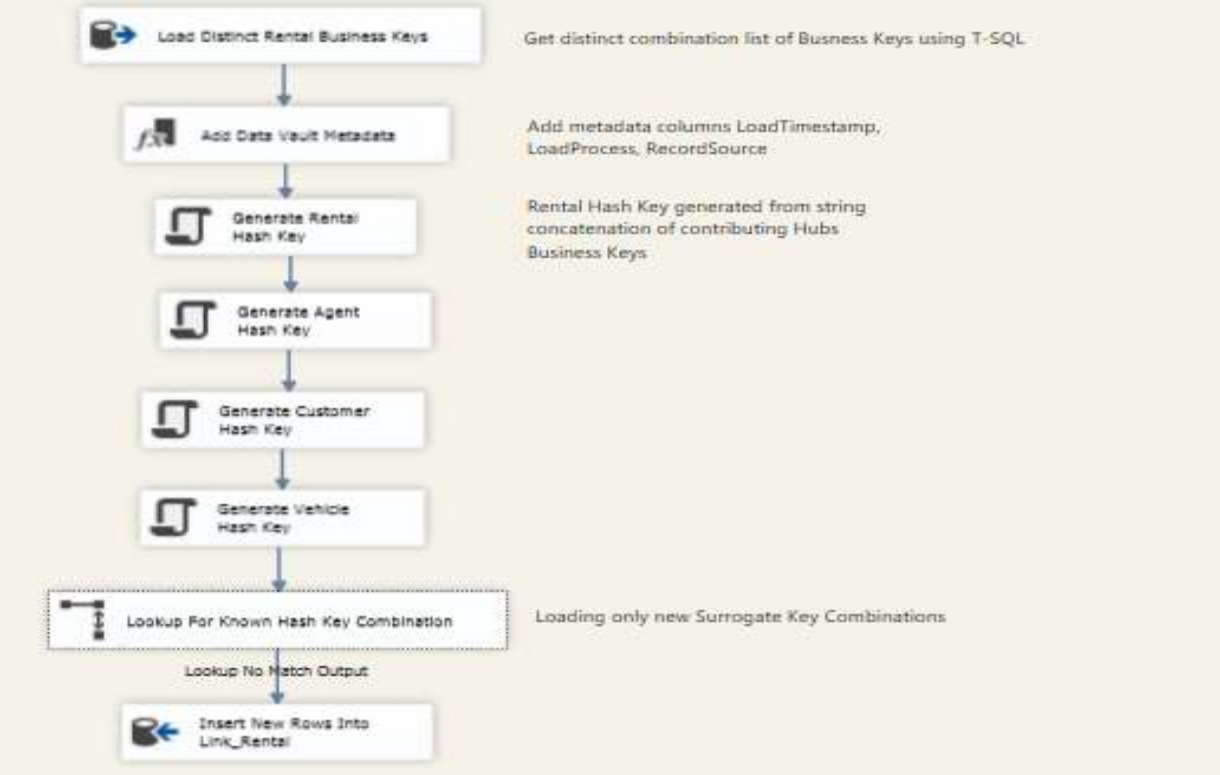
DV2_EDW_Load.dtsx

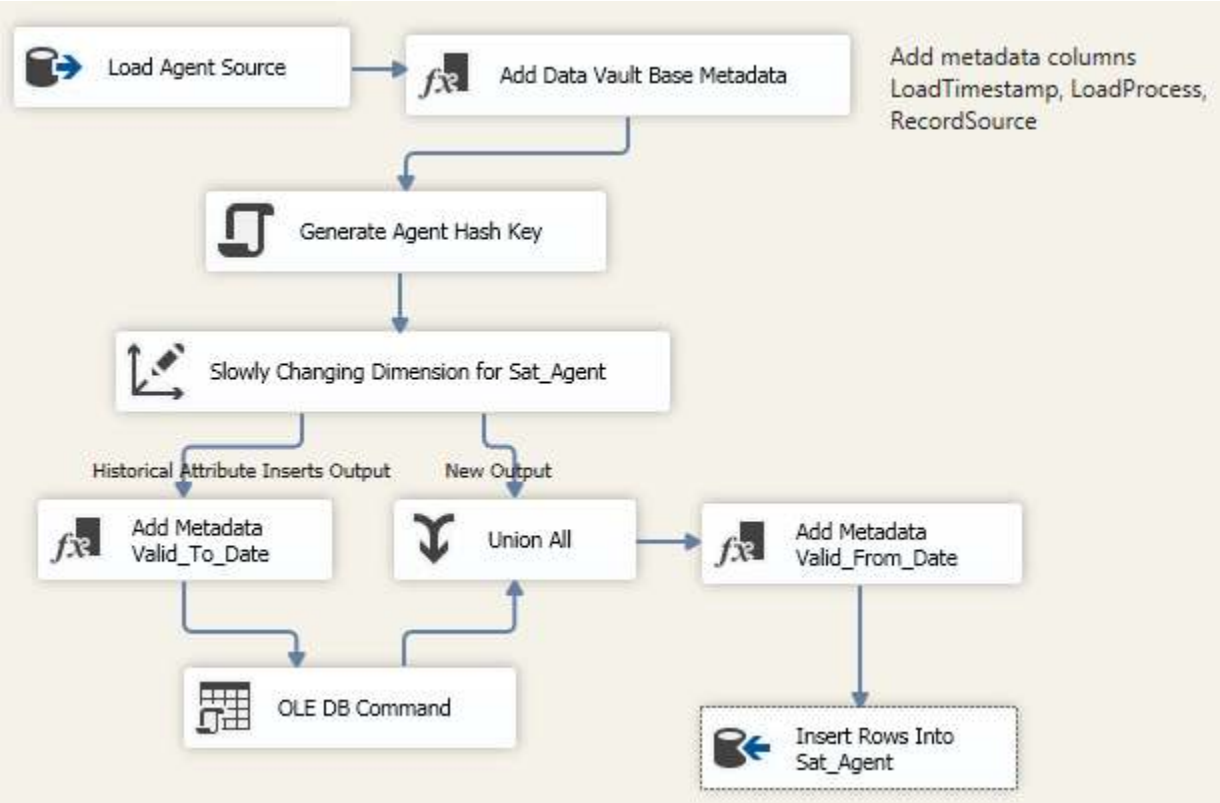


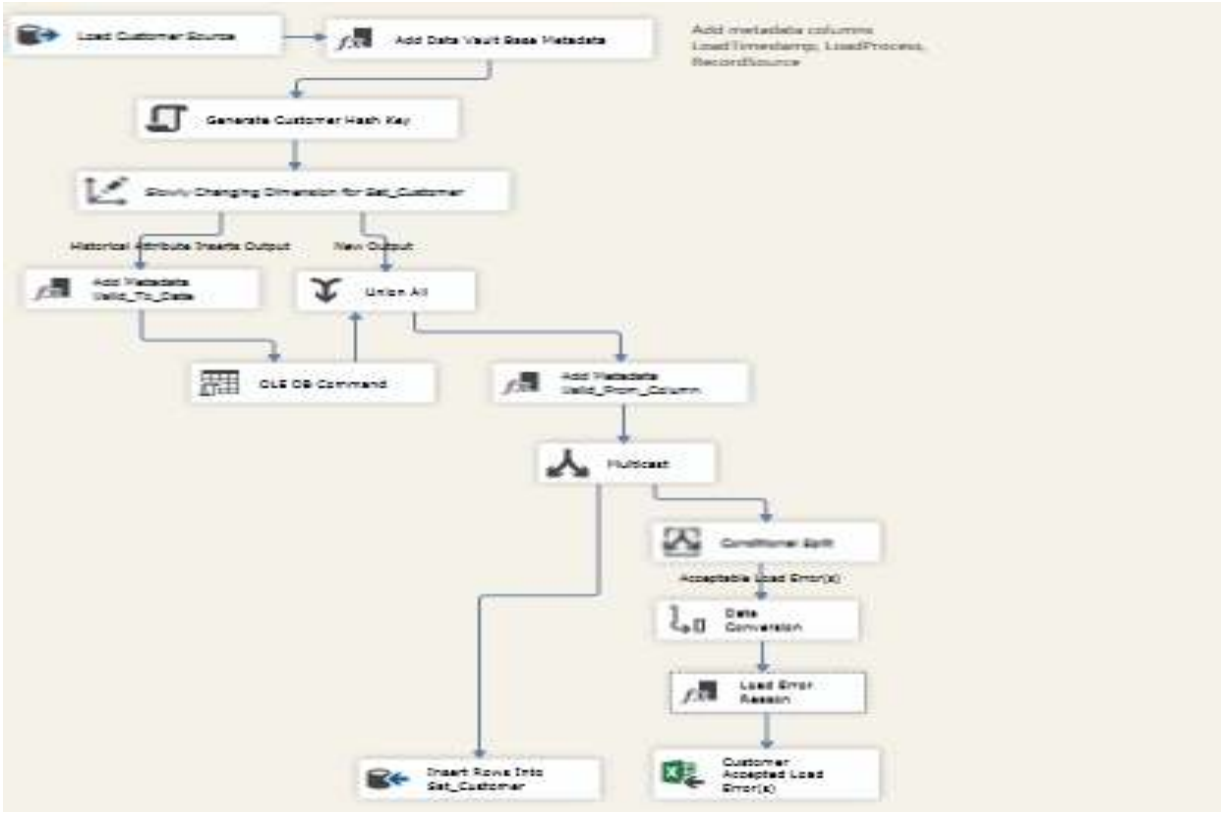


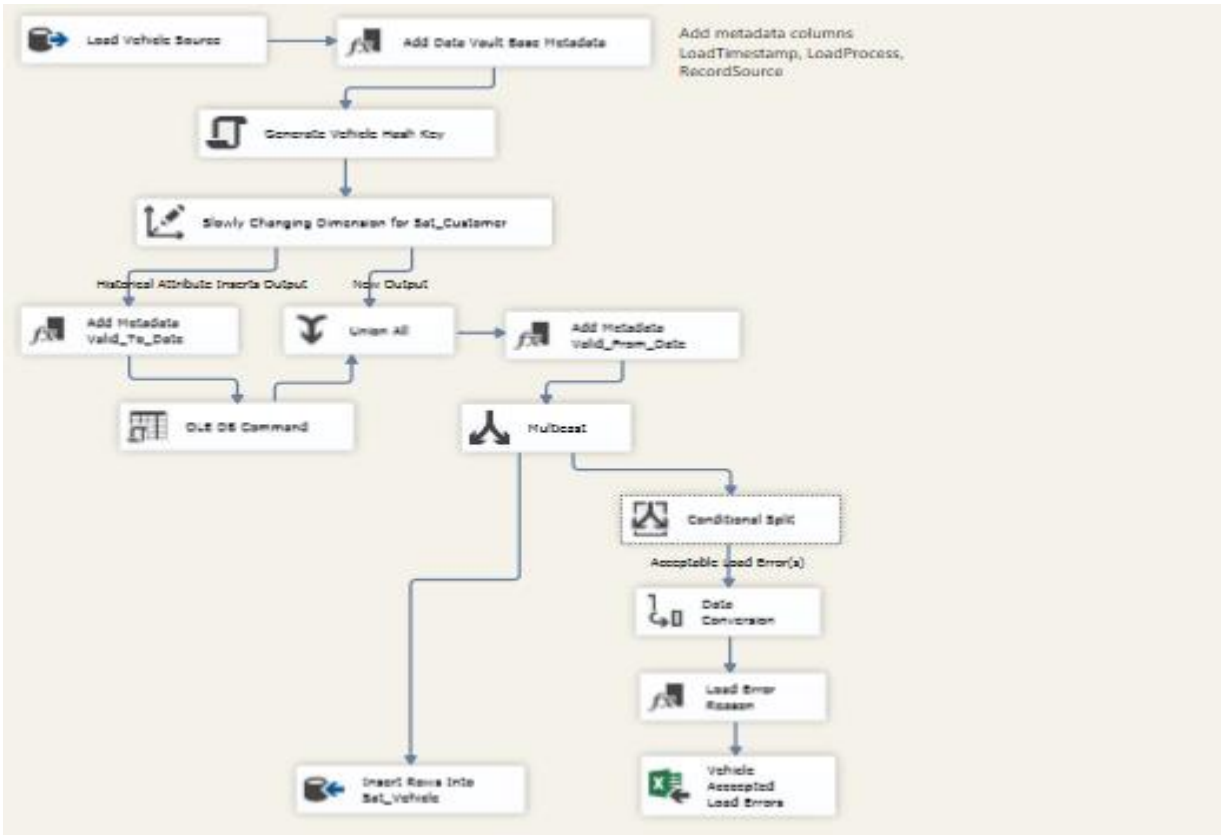

Load Distinct Customer Business Keys
Get distinct list of Business Keys using T-SQL
Add Data Vault Metadata
Add metadata columns LoadTimestamp, LoadProcess, RecordSource
Lookup For Known Business Keys
Lookup to select only new Business Keys (existing ones are simply ignored)
Lookup No Match Output
Generate Customer Hash Key
Generate Hask Key from Business Key
Insert New Rows Into Hub_Customer
Insert new Customer records

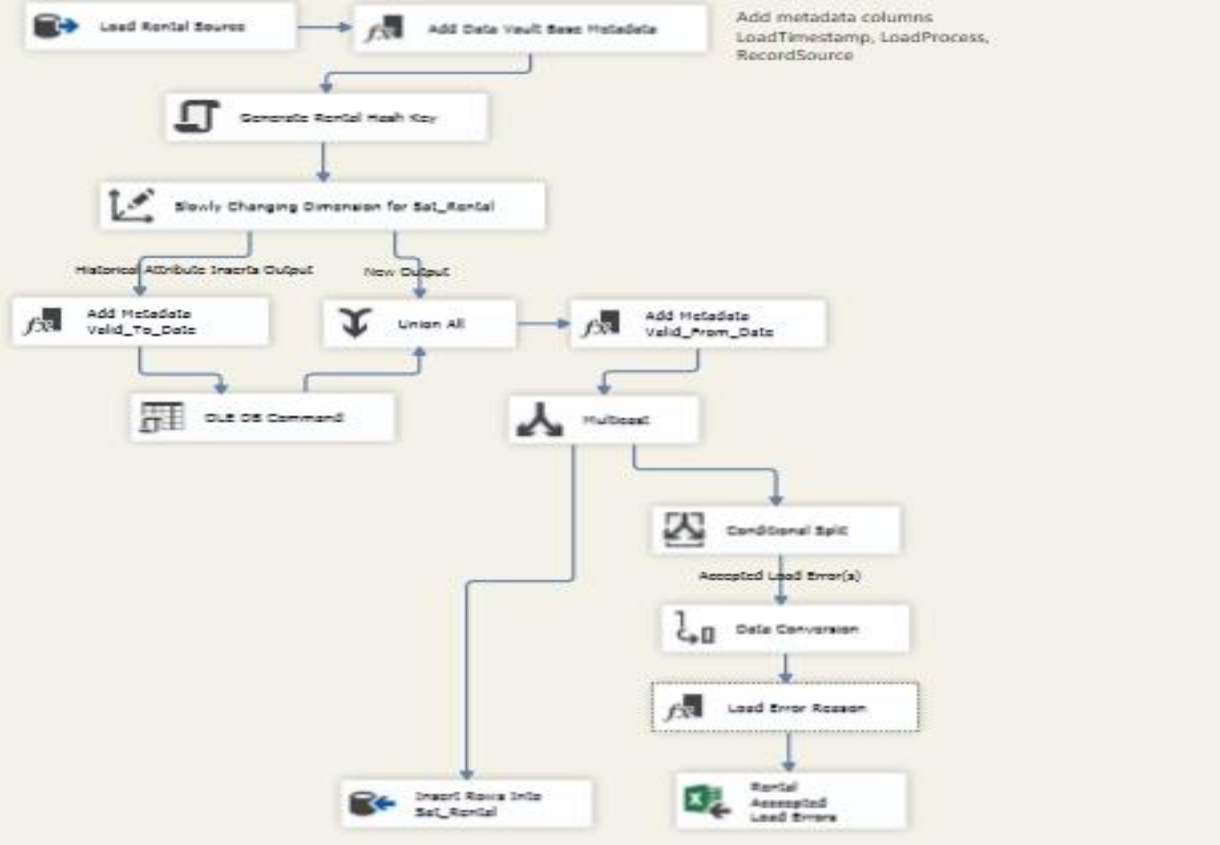

Load Distinct Vehicle Business Keys
Get distinct list of Business Keys using T-SQL
Add Data Vault Metadata
Add metadata columns LoadTimestamp, LoadProcess, RecordSource
Lookup For Known Business Keys
Lookup to select only new Business Keys (existing ones are simply ignored)
Lookup No Match Output
Generate Hash Key
Generate Hask Key from Business Key
Insert New Rows Into Hub_Vehicle
Insert new Agent records


Load Distinct Rental Business Keys
Get distinct combination list of Busness Keys using T-SQL
Add Data Vault Metadata
Add metadata columns LoadTimestamp, LoadProcess, RecordSource
Generate Rental Hash Key
Rental Hash Key generated from string concatenation of contributing Hubs Business Keys
Generate Agent Hash Key
Generate Customer Hash Key
Generate Vehicle Hash Key
Lookup For Known Hash Key Combination
Loading only new Surrogate Key Combinations
Lookup No Match Output
Insert New Rows Into Link_Rental

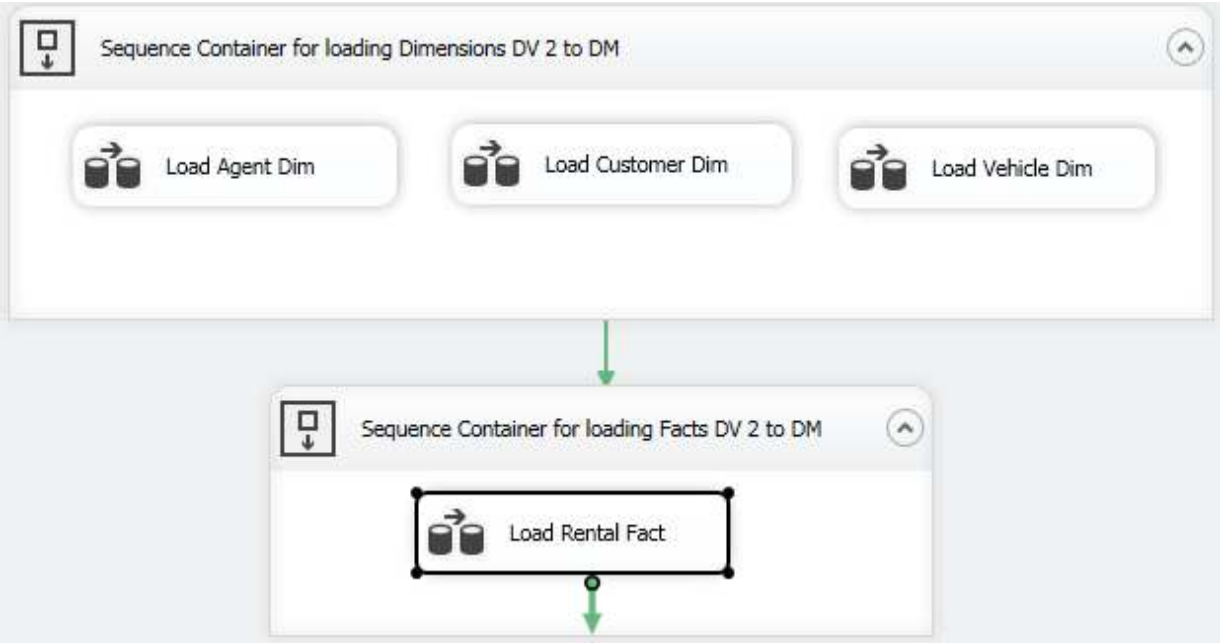

Load Agent Source
Add Data Vault Base Metadata
Add metadata columns LoadTimestamp, LoadProcess, RecordSource
Generate Agent Hash Key
Slowly Changing Dimension for Sat_Agent
Historical Attribute Inserts Output
New Output
Add Metadata Valid_To_Date
Union All
Add Metadata Valid_From_Date
OLE DB Command
Insert Rows Into Sat_Agent

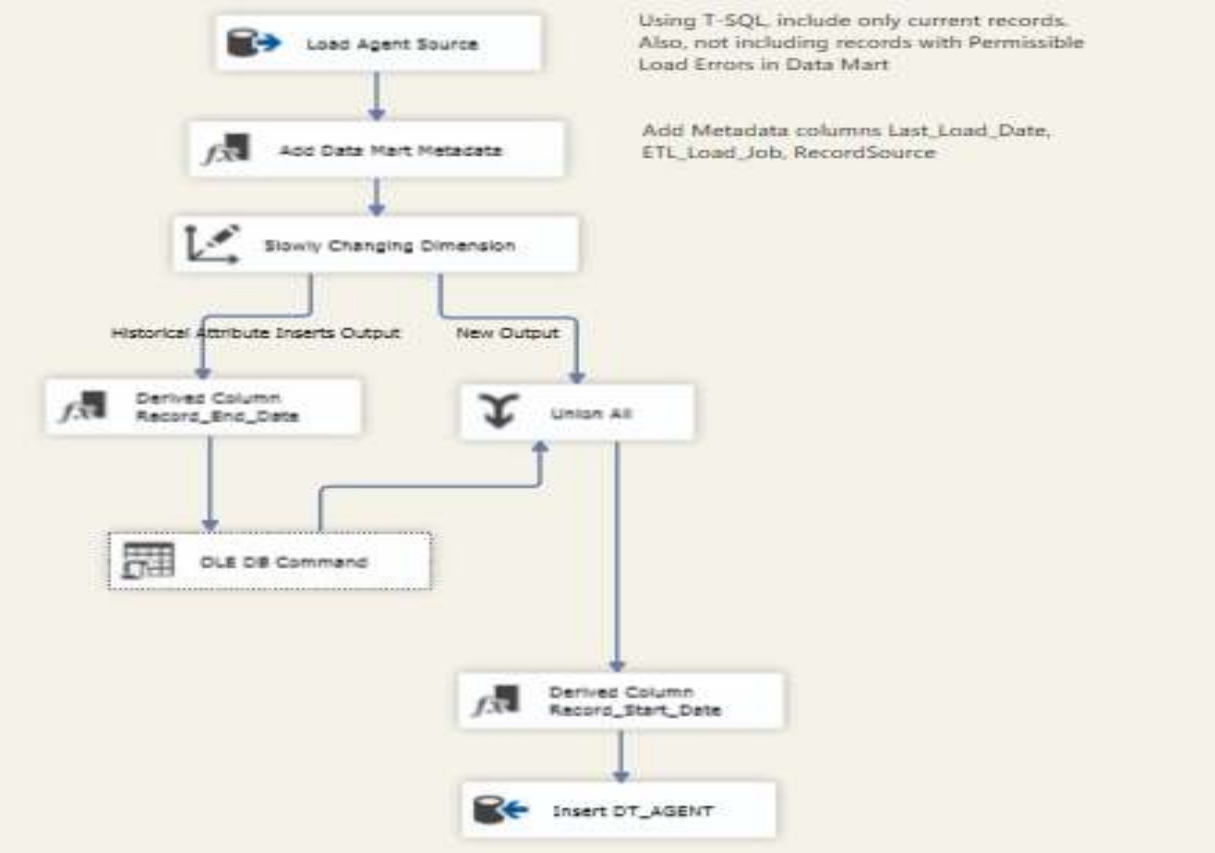

Load Customer Source
Add Data Vault Base Metadata
Add metadata columns
LoadTimestamp, LoadProcess,
RecordSource
Generate Customer Hash Key
Slowly Changing Dimension for Sat_Customer
Historical Attribute Inserts Output
New Output
Add Metadata
Valid_To_Date
Union All
OLE DB Command
Add Metadata
Valid_From_Column
Multicast
Conditional Split
Acceptable Load Error(s)
Data
Conversion
Load Error
Reason
Insert Rows Into
Sat_Customer
Customer
Accepted Load
Error(s)

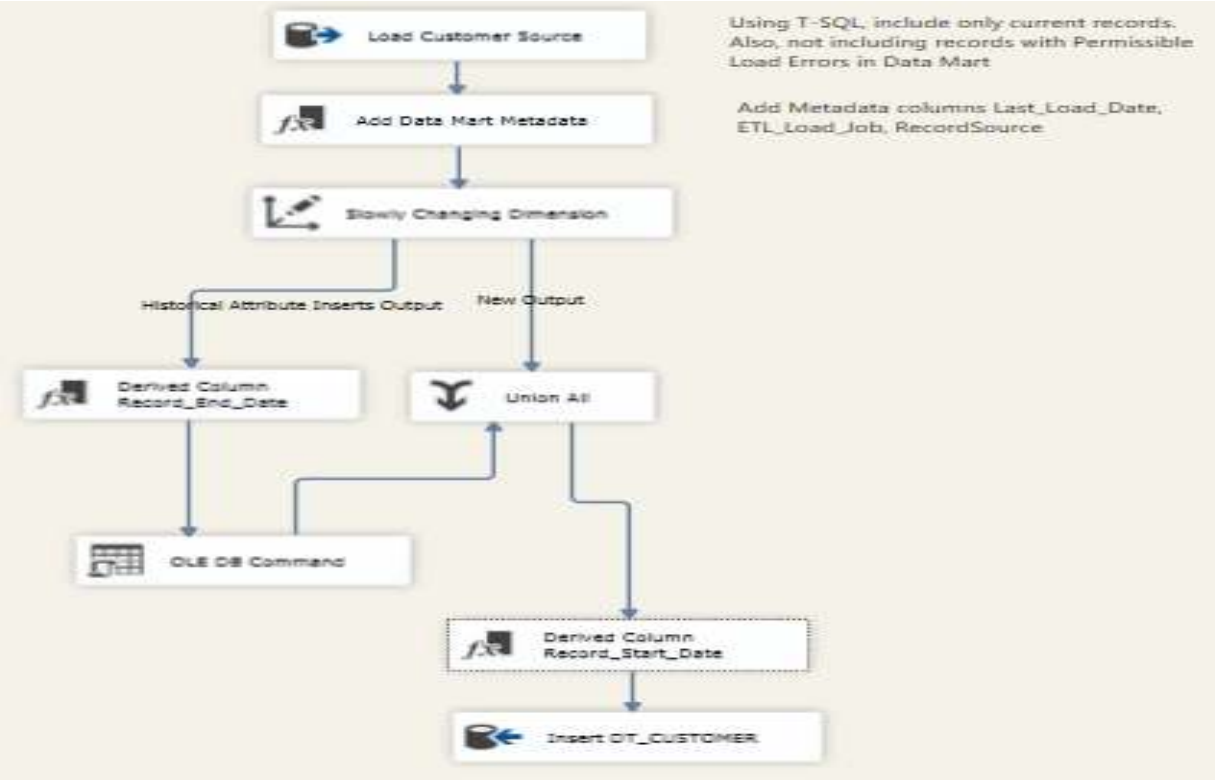

Load Vehicle Source
Add Data Vault Base Metadata
Add metadata columns
LoadTimestamp, LoadProcess,
RecordSource
Generate Vehicle Hash Key
Slowly Changing Dimension for Sat_Customer
Historical Attribute Inserts Output
New Output
Add Metadata
Valid_To_Date
Union All
Add Metadata
Valid_From_Date
OLE DB Command
Multicast
Conditional Split
Acceptable Load Error(s)
Data
Conversion
Load Error
Reason
Insert Rows Into
Sat_Vehicle
Vehicle
Accepted
Load Errors

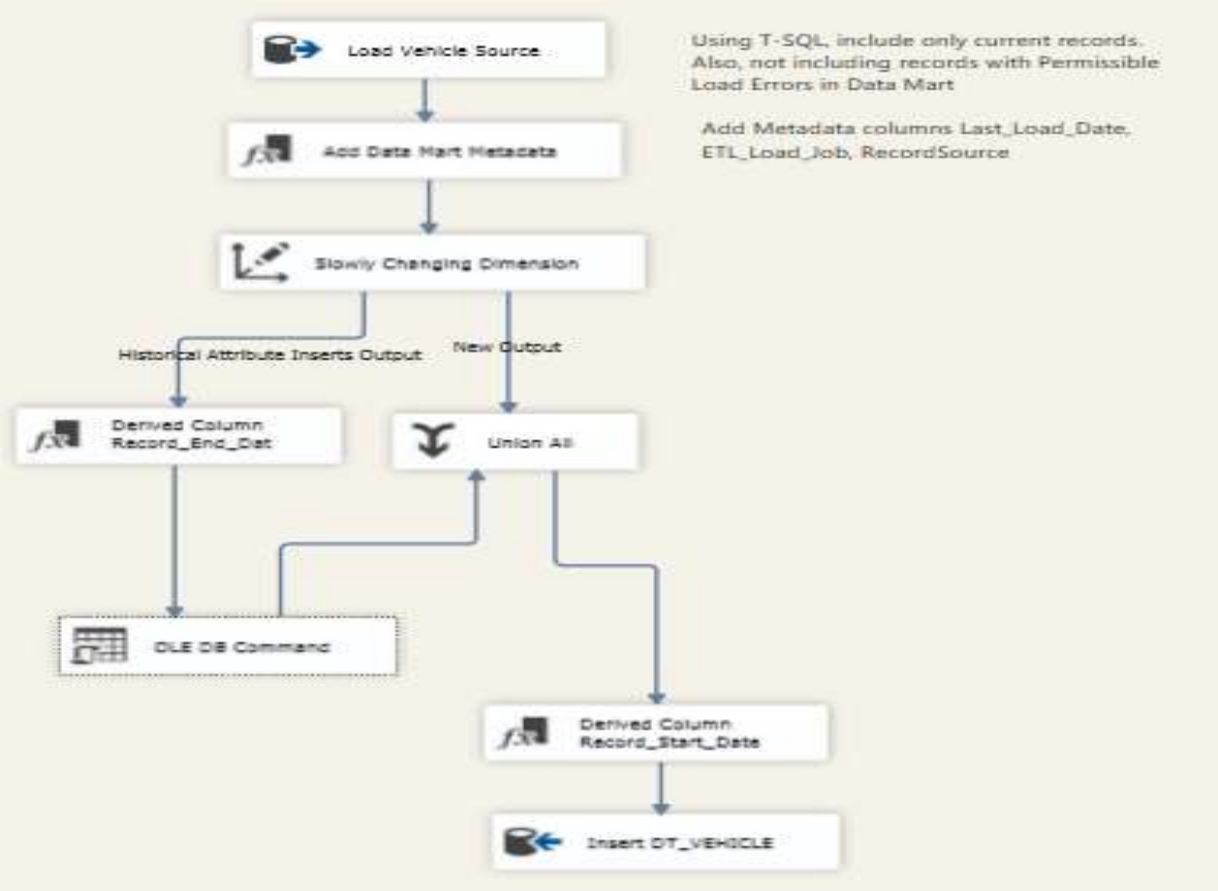


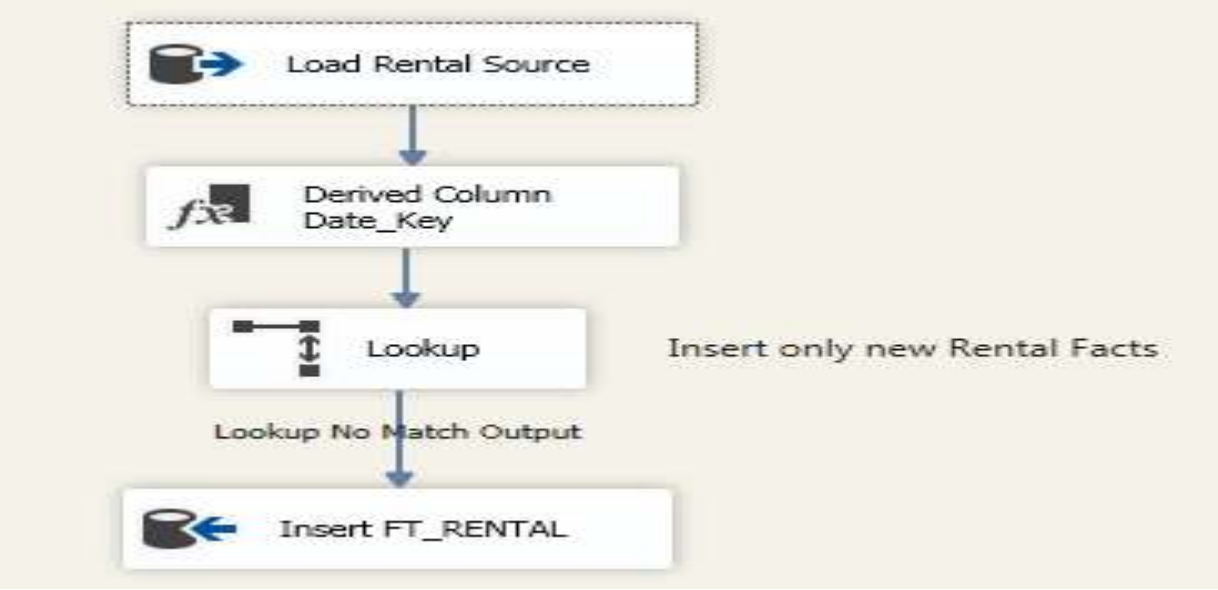
DV2_DM_Load.dtsx




Load Agent Source
Using T-SQL, include only current records. Also, not including records with Permissible Load Errors in Data Mart
Add Data Mart Metadata
Add Metadata columns Last_Load_Date, ETL_Load_Job, RecordSource
Slowly Changing Dimension
Historical Attribute Inserts Output
New Output
Derived Column Record_End_Date
Union All
OLE DB Command
Derived Column Record_Start_Date
Insert DT_AGENT


Load Customer Source
Using T-SQL, include only current records. Also, not including records with Permissible Load Errors in Data Mart
Add Data Mart Metadata
Add Metadata columns Last_Load_Date, ETL_Load_Job, RecordSource
Slowly Changing Dimension
Historical Attribute Inserts Output
New Output
Derived Column Record_End_Date
Union All
OLE DB Command
Derived Column Record_Start_Date
Insert DT_CUSTOMER


Load Vehicle Source
Using T-SQL, include only current records. Also, not including records with Permissible Load Errors in Data Mart
Add Data Mart Metacata
Add Metadata columns Last_Load_Date, ETL_Load_Job, RecordSource
Slowly Changing Dimension
Historical Attribute Inserts Output
New Output
Derived Column Record_End_Dat
Union All
OLE DB Command
Derived Column Record_Start_Date
Insert DT_VEHICLE


Load Rental Source
Derived Column Date_Key
Lookup
Insert only new Rental Facts
Lookup No Match Output
Insert FT_RENTAL