# Model Driven Software Engineering for Data Warehousing — Part 2: Data Warehousing

Simplicity is the ultimate sophistication.

— Leonardo da Vinci

This article is the second part of a series of articles in which I want to give an overview of how I think Model Driven Software Engineering (MDSE) can be used for data warehousing.

This article is about the creation of a means to store historical data by means of model to model and model to code transformations. I’ll explain how the historical data is stored and how the transformations work with code examples. I’ll also share some afterthoughts on the process and what I left out in this article for simplicity.

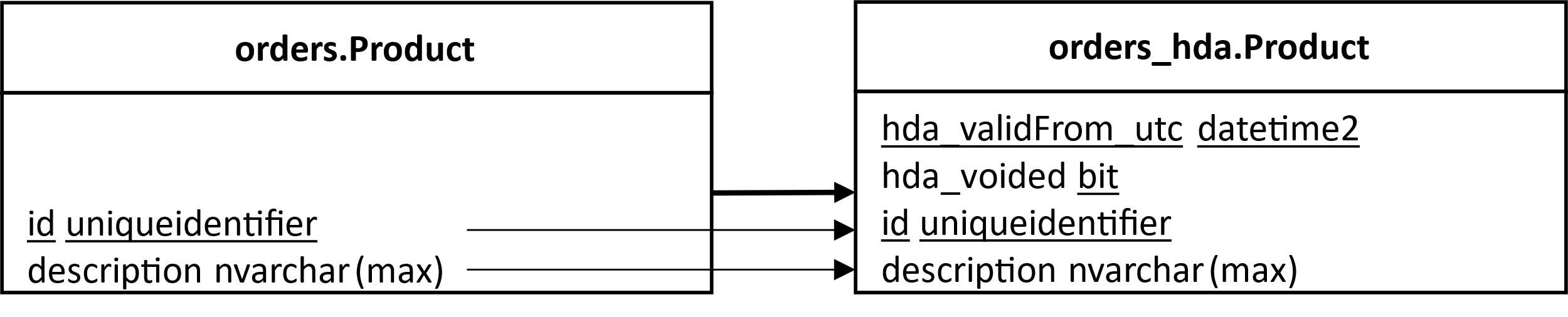
This post contains some code snippets, if you are interested in the full code, you can find in on GitHub.

* Python code can be found here:  
  <https://github.com/Maarten-gh/mdsedwhblog/blob/main/src/article-02.py>.
* Templates can be found here:  
  <https://github.com/Maarten-gh/mdsedwhblog/tree/main/article-02/templates>
* Generated code can be found here:  
  <https://github.com/Maarten-gh/mdsedwhblog/tree/main/article-02/code>

## What is an HDA

In this article I discuss the generation of HDA to store history about changes in source data. The steps for generation are described below, but before we can start generating, a basic understanding of an HDA is required. An HDA consists of a set of HDA tables. each HDA table is a means of storing changes to records in a source table.

Each record in the source table is identified by the values of its primary key columns. This means that for every record at most one instance can exists at any time in the source table. The HDA table accompanies contains all columns of its data table, and ads a timestamp column. A timestamp column is added to the HDA table. The timestamp column is called hda\_validFrom\_utc, as it stores the point in time when a record change was detected. The primary key of the HDA table consists of the primary key columns of the source table together with the new timestamp column. We are also interested in knowing when a record was removed from the source table, so a hda\_voided column is added to the HDA table as well. If a record that was present in the source table is removed at one point in time, a row will be added to the HDA with the primary key of the missing record and a 1 as value for hda\_voided. The image below shows an example of how a HDA table is related to a source table.



Now that the structure of the HDA table is known, we have to consider how source data is loaded into the HDA table. Loading the table can occur at set intervals. This happens in two steps.

1. Records from the source table are added to the HDA table together with the timestamp when the load started, but only if the HDA does not contain a record with the same key as the source record or if there exists such a record, but one or more of its properties changed.
2. After all new and changed records are inserted, the records in the HDA table are compared to the records in the source table to determine whether HDA records exists that are not in the source table. In that case, a new record is inserted into the HDA table with the key of the missing record and a 1 for hda\_voided.

After all this theoretical matter, an example might make things more clear. Assume we have 3 products in our Product table (the table on the left). On December 1st, we load these products in our HDA table (the table on the right), basically resulting in a copy of our source data enriched with the time of loading.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **id** | **description** |  |  | **hda\_validFrom\_utc** | **hda\_voided** | **id** | **description** |
| 1 | Red hat |  |  | 1-12-2023 | 0 | 1 | Red hat |
| 2 | Green glasses |  |  | 1-12-2023 | 0 | 2 | Green glasses |
| 3 | Blue jeans |  |  | 1-12-2023 | 0 | 3 | Blue jeans |

Over the course of time, we notice that the green glasses are actually yellow and our blue jeans are sold. These changes are represented in the source table. On December 5th we load the HDA table again. The red hat was not changed, so no new row was added. New rows are added for our changed glasses and removed jeans. Notice the hda\_voided value for the jeans is 1.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **id** | **description** |  |  | **hda\_validFrom\_utc** | **hda\_voided** | **id** | **description** |
| 1 | Red hat |  |  | 1-12-2023 | 0 | 1 | Red hat |
| 2 | Yellow glasses |  |  | 1-12-2023 | 0 | 2 | Green glasses |
|  |  |  |  | 5-12-2023 | 0 | 2 | Yellow glasses |
|  |  |  |  | 1-12-2023 | 0 | 3 | *NULL* |
|  |  |  |  | 5-12-2023 | 1 | 3 | Blue jeans |

A few days later, we notice that the glasses are actually chartreuse, so we change it again in the source table. Moreover, the blue jeans are returned, so we add them back to our inventory. On December 9th, we load our HDA again. New rows are added again for the classes and the jeans, indicating that a change was detected at this date.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **id** | **description** |  |  | **hda\_validFrom\_utc** | **hda\_voided** | **id** | **description** |
| 1 | Red hat |  |  | 1-12-2023 | 0 | 1 | Red hat |
| 2 | Chartreuse glasses |  |  | 1-12-2023 | 0 | 2 | Green glasses |
| 3 | Blue jeans |  |  | 5-12-2023 | 0 | 2 | Yellow glasses |
|  |  |  |  | 9-12-2023 | 0 | 2 | Chartreuse glasses |
|  |  |  |  | 1-12-2023 | 0 | 3 | Bluejeans |
|  |  |  |  | 5-12-2023 | 1 | 3 | *NULL* |
|  |  |  |  | 9-12-2023 | 0 | 3 | Blue jeans |

The HDA several mildly interesting properties.

* All changes of source data are recorded, but only when required. If properties of a record do not change, then no new row is inserted into the HDA.
* All changes are done with INSERT statements and should be fast.
* From the point in time when a record from the source table enters the HDA, there will always be a record present for that record, even if it is deleted.

## The Ultimate Goal

The ultimate goal of our endeavour is to generate DDL code for our HDA tables. This is not very exciting as we already accomplished this in the previous blog. But, for the sake of completeness, below is the SQL code for the Product HDA table that we expect. Note that null values are allowed for all source columns that are not part of the primary key.

1. **CREATE** **TABLE** [orders\_hda].[Product] (
2. [hda\_validFrom\_utc] datetime2 NOT **NULL**
3. , [hda\_voided] bit NOT **NULL**
4. , [id] uniqueidentifier NOT **NULL**
5. , [description] nvarchar(255) **NULL**
6. , **CONSTRAINT** [pk\_Product]
7. **PRIMARY** **KEY** (
8. [id]
9. , [hda\_validFrom\_utc]
10. )
11. )
12. ;
13. **GO**

Next, it would be nice to have a Point in Time (PIT) function. The function takes a timestamp as input and returns the state of the source at that point in time. This function has two main goals. The first one is to give the users of our HDA the possibility to travel back in time without all the hassle! And the second one is that we can used it in the ETL code that we generate, as you will see shortly.

1. -- Create PIT function for [orders\_hda].[Product]
2. **CREATE** OR **ALTER** **FUNCTION** [orders\_hda].[ufn\_pit\_Product] (
3. @timestamp\_utc datetime2
4. )
5. RETURNS **TABLE**
6. **AS**
7. **RETURN** (
8. **SELECT**
9. [hda\_validFrom\_utc]
10. , [hda\_voided]
11. , [id]
12. , [description]
13. **FROM**
14. [orders\_hda].[Product] **AS** [d]
15. **WHERE**
16. [d].[hda\_validFrom\_utc] <= @timestamp\_utc
17. AND
18. [d].[hda\_voided] = 0
19. AND
20. [hda\_validFrom\_utc] = (
21. **SELECT**
22. MAX([hda\_validFrom\_utc])
23. **FROM**
24. [orders\_hda].[Product] **AS** [h]
25. **WHERE**
26. [d].[id] = [h].[id]
27. AND
28. [h].[hda\_validFrom\_utc] <= @timestamp\_utc
29. )
30. )

Last but certainly not least, we want to generate ETL code to populate our HDA tables from their source tables. This is done in two steps. First, we take all new and changed records from the source tables and add them to our HDA table. Note the values for hda\_validFrom\_utc and hda\_voided, respectively the value of the parameter @timestamp\_utc and 0. After we added all changes are added, we compare the latest state of the HDA table to the source table. If there are records in the HDA table that are not in the source table then new records are added to the HDA table with value 1 for hda\_voided, a value of the primary key and all other columns NULL as shown in the examples in the previous section.

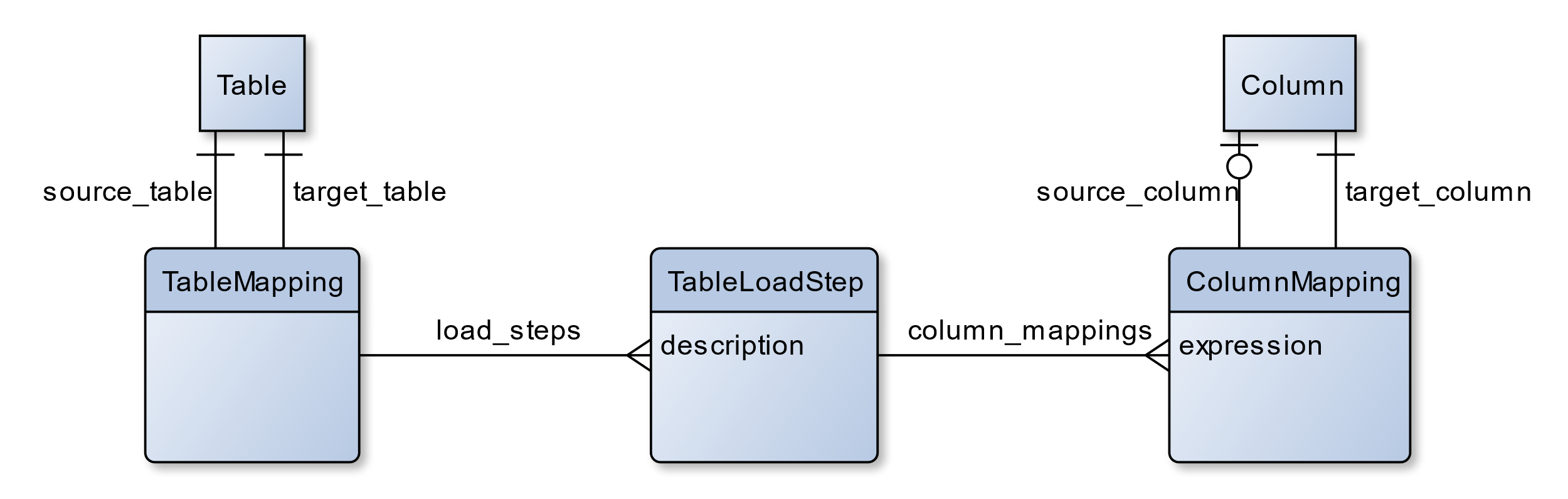
1. **CREATE** OR **ALTER** **PROCEDURE** [orders\_hda].[usp\_load\_Product] (
2. @timestamp\_utc datetime2
3. )
4. **AS**
5. **BEGIN**
6. -- Load HDA table with changes from source table.
7. **INSERT** **INTO** [orders\_hda].[Product] (
8. [hda\_validFrom\_utc]
9. , [hda\_voided]
10. , [id]
11. , [description]
12. )
13. **SELECT**
14. @timestamp\_utc
15. , 0
16. , [id]
17. , [description]
18. **FROM**
19. [orders].[Product] **AS** [src]
20. **WHERE** NOT EXISTS (
21. **SELECT**
22. 1
23. **FROM**
24. [orders\_hda].[ufn\_pit\_Product](@timestamp\_utc) **AS** [hda]
25. **WHERE**
26. 1=1
27. AND
28. [src].[id] = [hda].[id]
29. AND
30. ([src].[description] = [hda].[description]
31. OR ([src].[description] **IS** **NULL** AND [hda].[description] **IS** **NULL**))
33. )
34. ;
35. -- Add records to the HDA table for deleted source records.
36. **INSERT** **INTO** [orders\_hda].[Product] (
37. [hda\_validFrom\_utc]
38. , [hda\_voided]
39. , [id]
40. )
41. **SELECT**
42. @timestamp\_utc
43. , 1
44. , [id]
45. **FROM**
46. [orders\_hda].[ufn\_pit\_Product](@timestamp\_utc) **AS** [hda]
47. **WHERE** NOT EXISTS (
48. **SELECT**
49. 1
50. **FROM**
51. [orders].[Product] **AS** [src]
52. **WHERE**
53. 1=1
54. AND
55. [src].[id] = [hda].[id]
56. )
57. ;
58. **END**
59. ;

## Generating the HDA model

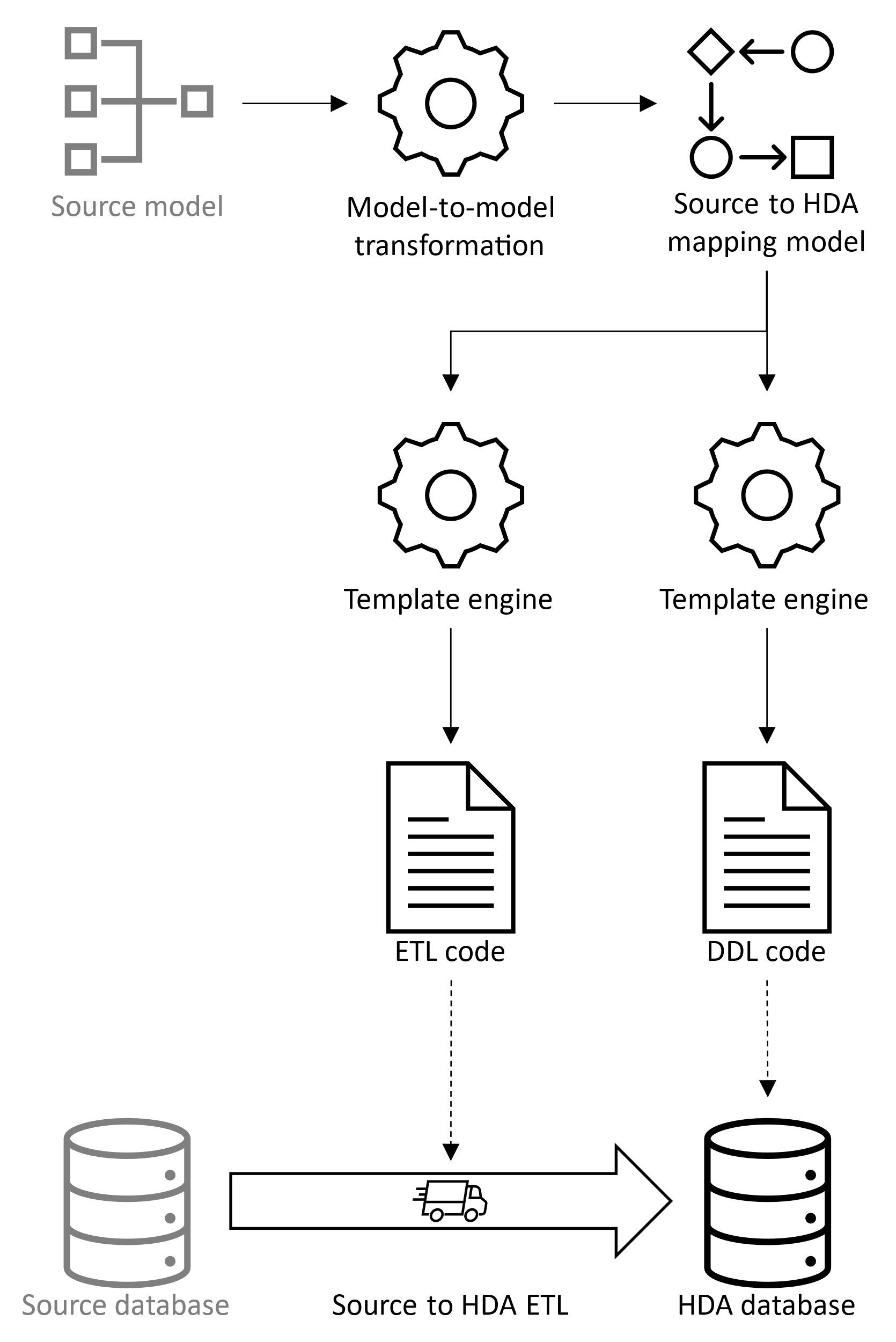
We have seen an example of an HAD table and how the SQL statements to create it and how to populate it look. For one table this process is simple, but consider the case where you would want to do this for multiple source systems consisting of hundreds of tables. Doing this manually will be time consuming, error prone and bóóóring. Luckily this example lends itself perfectly to automation.

We are going to generate a source to HDA mapping model based on a source model. In this article I assume that the source model is given, in the next article I’ll dive deeper into how to obtain it. The mapping model describes the HDA tables and how data should be moved from source tables into HDA tables. Based on the mapping model, we are going to generate DDL code to create the HDA tables and PIT functions and the ETL code to load data into the HDA tables from the source tables.

Below is an overview of the most important mapping concepts.

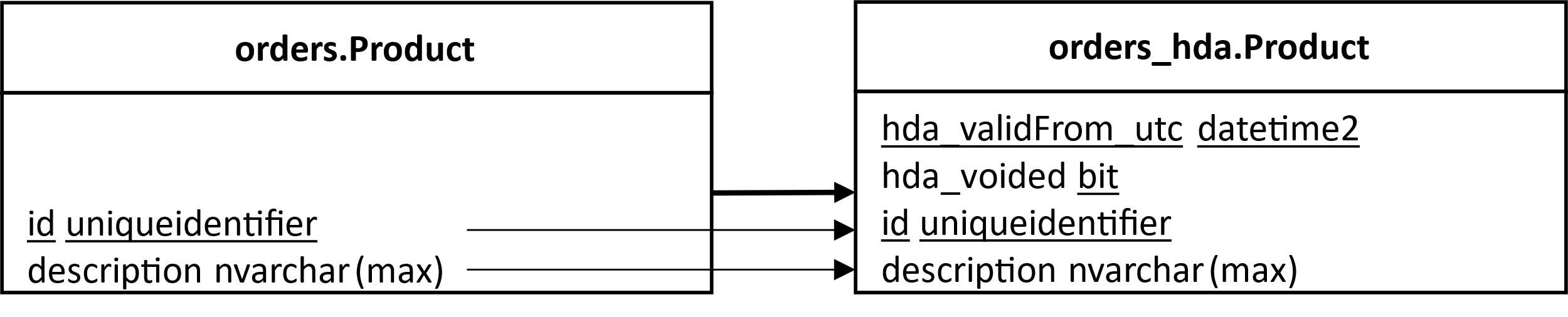


* ColumnMapping: This class sets the rules for moving data from one column to another, defining the optional source and target columns and any necessary transformation expression.
* TableLoadStep: Each step represents an action in the process of loading data into a table, accompanied by a description and a set of column mappings.
* TableMapping: Moving beyond individual columns, this class manages how entire tables connect, holding strategies for data transfer.



We’ll start from the model of the physical domain that I described in the previous article to model the source and HDA. This domain contains objects such as schemas, tables, columns and constraints. Moreover, an new mapping domain needs to be created to capture our data flow. We need mappings to define the relation between source schemas and HDA, source tables and HDA tables and source columns and HDA columns.

The mapping domain basically captures the arrows that are shown in the image below. Now we need some extra information to be able to fill the HDA table as shown in the previous section. For instance, we need to know when the value hda\_voided is 0 or 1 and we need to know how to populate hda\_validFrom\_utc.



The code for this looks a lot like the code from the previous post, so I’ll only show the most interesting parts here.

1. **def** transfrom\_source\_table\_to\_hda\_table\_mapping(
2. source\_table: Table
3. ) -> TableMapping:
4. technical\_hda\_column\_mappings = create\_technical\_hda\_column\_mappings()
5. hda\_column\_mappings = \
6. technical\_hda\_column\_mappings + \
7. [transform\_source\_column\_to\_hda\_column\_mapping(
8. c, source\_table.primary\_key\_constraint.column\_names
9. ) **for** c **in** source\_table.columns
10. ]
11. hda\_table = Table(
12. name=source\_table.name,
13. primary\_key\_constraint=create\_hda\_primary\_key\_constraint(
14. source\_table.primary\_key\_constraint),
15. columns=[m.target\_column **for** m **in** hda\_column\_mappings],
16. foreign\_key\_constraints=[],
17. )
18. mapping = TableMapping(
19. source\_table=source\_table,
20. target\_table=hda\_table,
21. column\_mappings=hda\_column\_mappings,
22. )
23. **return** mapping
24. **def** create\_technical\_hda\_column\_mappings() -> list[ColumnMapping]:
25. **return** [
26. ColumnMapping(
27. source\_column=**None**,
28. target\_column=Column(
29. name="hda\_validFrom\_utc",
30. datatype="datetime2",
31. nullable=**False**,
32. ),
33. expression='@timestamp\_utc'
34. ),
35. ColumnMapping(
36. source\_column=**None**,
37. target\_column=Column(
38. name="hda\_voided",
39. datatype="bit",
40. nullable=**False**,
41. ),
42. expression='0'
43. )
44. ]

## Generating HDA Code

Bla bla bla

1. -- Load HDA table with data from source table.
2. INSERT INTO {{ get\_hda\_table\_name(schema\_mapping, table\_mapping) }} (
3. {{ hda\_column\_name\_list(table\_mapping) -}}
4. )
5. SELECT
6. {{ hda\_column\_value\_list(table\_mapping, {}) -}}
7. FROM
8. {{ get\_source\_table\_name(schema\_mapping, table\_mapping) }} AS [src]
9. WHERE NOT EXISTS (
10. SELECT
11. 1
12. FROM
13. {{ get\_ufn\_pit\_name(schema\_mapping, table\_mapping) }}(@timestamp\_utc) AS [hda]
14. WHERE
15. 1=1
16. {% **for** column\_name **in** table\_mapping.source\_table.primary\_key\_constraint.column\_names -%}
17. AND
18. {{ compare\_source\_column\_to\_target\_column(column\_name, table\_mapping) }}
19. {% **endfor** -%}
20. {% **for** column **in** table\_mapping.source\_table.columns **if** column.name not **in** table\_mapping.source\_table.primary\_key\_constraint.column\_names -%}
21. AND
22. {{ null\_safe\_compare\_source\_column\_to\_target\_column(column.name, table\_mapping) }}
23. {% **endfor** -%}
24. )
25. -- Add records to the HDA table for deleted source records.
26. INSERT INTO {{ get\_hda\_table\_name(schema\_mapping, table\_mapping) }} (
27. {{ hda\_column\_name\_list(table\_mapping) -}}
28. )
29. SELECT
30. {{ hda\_column\_value\_list(table\_mapping, { "hda\_voided": 1 }) -}}
31. FROM
32. {{ get\_ufn\_pit\_name(schema\_mapping, table\_mapping) }}(@timestamp\_utc) AS [hda]
33. WHERE NOT EXISTS (
34. SELECT
35. 1
36. FROM
37. {{ get\_source\_table\_name(schema\_mapping, table\_mapping) }} AS [src]
38. WHERE
39. 1=1
40. {% **for** column\_name **in** table\_mapping.source\_table.primary\_key\_constraint.column\_names -%}
41. AND
42. {{ compare\_source\_column\_to\_target\_column(column\_name, table\_mapping) }}
43. {% **endfor** -%}
44. )

## Afterthoughts

One could argue that the historical storage described in this article is not optimal. And I would agree. The goal of this article was to show techniques and considerations when applying MDSE to automate generation of load procedures and I believe that the examples shown in this article are sufficient for that matter.

The mapping structure that I used in this post is rather straightforward. Using it for more complex cases requires a revision of the structure, and in extension of that, adaptions to the templates. Non the less, the techniques show with load types, steps and column expressions as used for this article are already quite powerful.

Besides generation of the rather technical SQL code that was discussed in this post we can use the same approach and mapping model to generate other artifacts such as documentation. For instance, a lineage diagram as shown in the figure below. This requires a template to generate Markdown code. Creating such a template is left as an exercise for the reader (or, for the lazy readers, it can be found in the GitHub repository).

Afbeelding met tekst, schermopname, diagram, lijn

Automatisch gegenereerde beschrijving