WHITE PAPER

Replicating and Transforming Semi-structured Data

Best Practices for Qlik Data Integration

Benjamin Perez-Goytia
Analytics Data Architect
Qlik

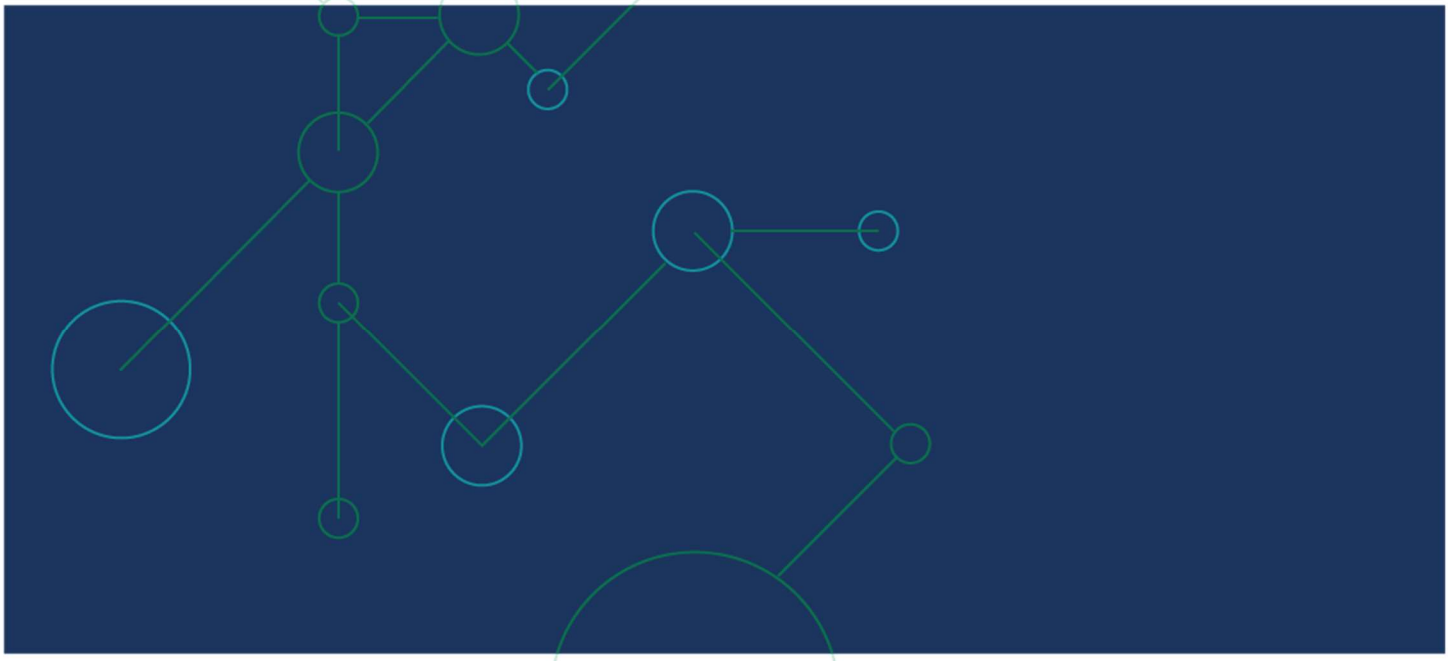


TABLE OF CONTENTS

SUMMARY

- Modern data warehouses often require replicating and transforming data that is both structured and semi-structured.
- Qlik Data Integration supports replicating and transforming both structured and semi-structured data.
- Qlik Compose for Data Warehouses is an EL-T tool, and it can take advantage of the built-in functions of the warehouse database to extract and transform semi-structured data.

INTRODUCTION

Today, most on-line transactional processing systems (OLTP) store data in a tabular format of tables and columns. In most cases, data is stored in SQL databases such as Microsoft SQL Server, PostgreSQL, Oracle, Snowflake and others, that support a variety of data types such as character, datetime, numeric, varchar, and more. Additionally, support for new data types such as Character Large Object (CLOB), Geography (GEO), JavaScript Object Notation (JSON) and Extensible Markup Language (XML) were added as databases evolved.

However, some of these new data types that have a semi-structured format don't obey the data structure traditionally associated with relational theory. For example, a column of type JSON or XML can be used to store both complex documents or a list of name-value pairs.

Consequently, you must transform semi-structured data to a tabular format when the intended use is for analytics or business intelligence.

This whitepaper discusses how to convert and consume JSON semi-structured data with the Qlik Data Integration solution.

Qlik Replicate

Identifying Semi-Structured Columns from the Source Table

The first step to replicating, transforming, and consuming semi-structured data is to identify the semi-structured columns to be replicated from the source database. Figure 1 below shows the structure of a PostgreSQL table called SRC_CUSTOMER. This table is used to store customer information and is the source for a customer dimension of a data warehouse.

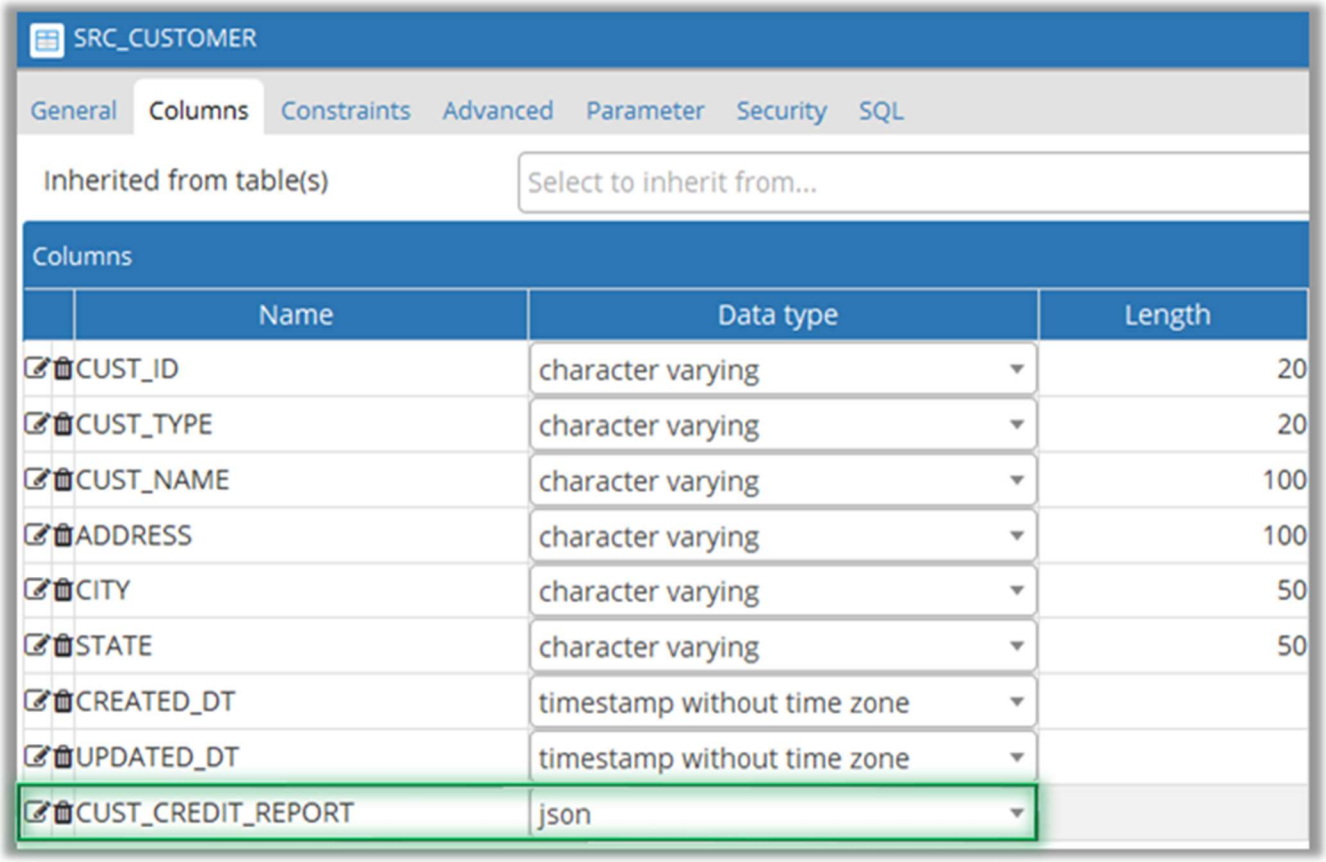


Figure 1. An example PostgreSQL table with a semi-structured column.

Notice that the table has a semi-structured column called CUST_CREDIT_REPORT which contains the credit report information for each customer. The column data type is JSON and the credit report is

stored as a JSON document. The following figure illustrates an example of a credit report document in JSON format.

CUST_CREDIT_REPORT


```
{
    "consumerPii": {
        "primaryApplicant": {
            "driverslicense": {
                "state": "CA",
                "number": "1234567"
            },
            "name": {
                "middleName": "QUINCY",
                "lastName": "CONSUMER",
                "firstName": "JONATHAN",
                "generationCode": "Sr"
            },
            "dob": {
                "dob": "3021945.0"
            },
            "phone":
                {
                    "type": "R",
                    "number": "818.555.1111"
                }
            ,
            "ssn": {
                "ssn": "999999990"
            },
            "employment": {
                "employerAddress": {
                    "city": "BURBANK",
                    "state": "CA",
                    "line2": " ",
                    "line1": "10655 N BIRCH ST",
                    "zipCode": "915021234"
                },
                "employerName": "HARDWARE STORE"
            }
        }
    }
}
```


Figure 2 - Example JSON document.

The structure of a JSON document consists of attribute-value pairs and array data types. The JSON document above has several individual attributes such as `dob` and `ssn`. There are also several attributes such as `driverslicense`, `name`, and `phone` that contain sub-attributes. The `driverslicense` attribute contains two additional sub-attributes: `state`, and `number`. Likewise, the attribute called `phone` contains two additional sub-attributes: `type`, and `number`.

Replicating Semi-Structured Data into a Target Table

The next step is to use Qlik Replicate to copy JSON-type columns from the source table into a target or landing table. Users must first create a task when using Qlik Replicate to copy structured or semi-structured data from a source table to a target table. Each task contains three components:

1. A source endpoint (i.e. a source connection)
2. A target endpoint (i.e. a connection to a data target)
3. A list of source tables to be replicated.

When the task executes, Qlik Replicate loads the source data into memory and converts the data into its own data types. It then maps its data types to the column types of the target database. Next it creates the target table, and finally loads the data into the target. Qlik Replicate returns an error for data that it cannot convert or load. Figure 3, highlights how Qlik Replicate maps the columns of the customer table, MY_SRC_AREA.SRC_CUSTOMER, into its own data types:

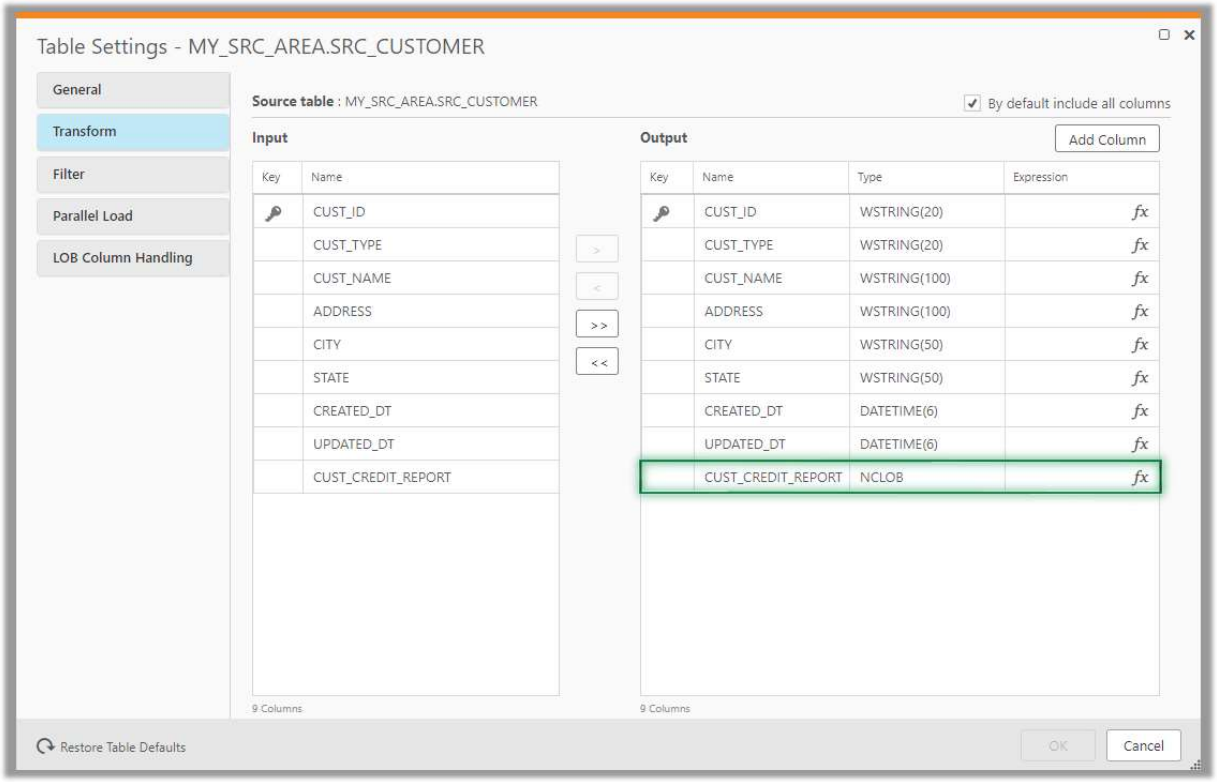


Figure 3 - Qlik Replicate task using an NCLOB column.

In the example above, Qlik Replicate converts column CUST_CREDIT_REPORT from JSON to NCLOB. JSON is the source column data type and NCLOB is the Qlik Replicate data type. Figure 4 illustrates how Qlik Replicate created BPG_DW.ORDERS.SRC_CUSTOMER table in a Snowflake target.

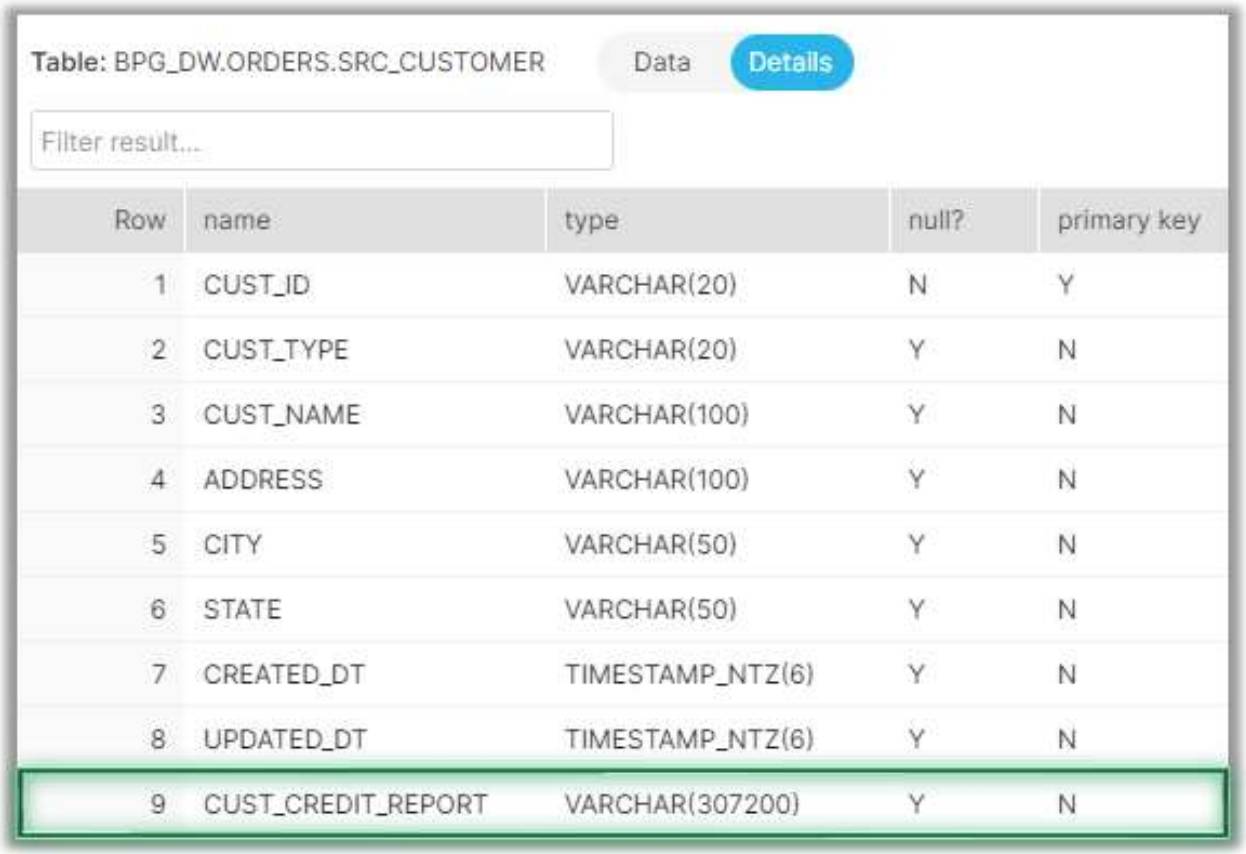


Table: BPG_DW.ORDERS.SRC_CUSTOMER Data Details

Filter result...

Row	name	type	null?	primary key
1	CUST_ID	VARCHAR(20)	N	Y
2	CUST_TYPE	VARCHAR(20)	Y	N
3	CUST_NAME	VARCHAR(100)	Y	N
4	ADDRESS	VARCHAR(100)	Y	N
5	CITY	VARCHAR(50)	Y	N
6	STATE	VARCHAR(50)	Y	N
7	CREATED_DT	TIMESTAMP_NTZ(6)	Y	N
8	UPDATED_DT	TIMESTAMP_NTZ(6)	Y	N
9	CUST_CREDIT_REPORT	VARCHAR(307200)	Y	N

Figure 4 - Snowflake table with a large VARCHAR column.

Figure 4 highlights how Qlik Replicate automatically created and mapped column CUST_CREDIT_REPORT from NCLOB to Snowflake's VARCHAR data type. Large columns such as NCLOB columns in Qlik Replicate are known internally as LOB columns. Also note the length of the target column. Qlik Replicate intelligently calculated the required length: 307200. However, the calculated column length in some cases may impact data transfer speeds. As a result, Qlik Replicate has a user-defined optimization that limits the size in bytes of LOB columns. This option is illustrated in Figure 5.

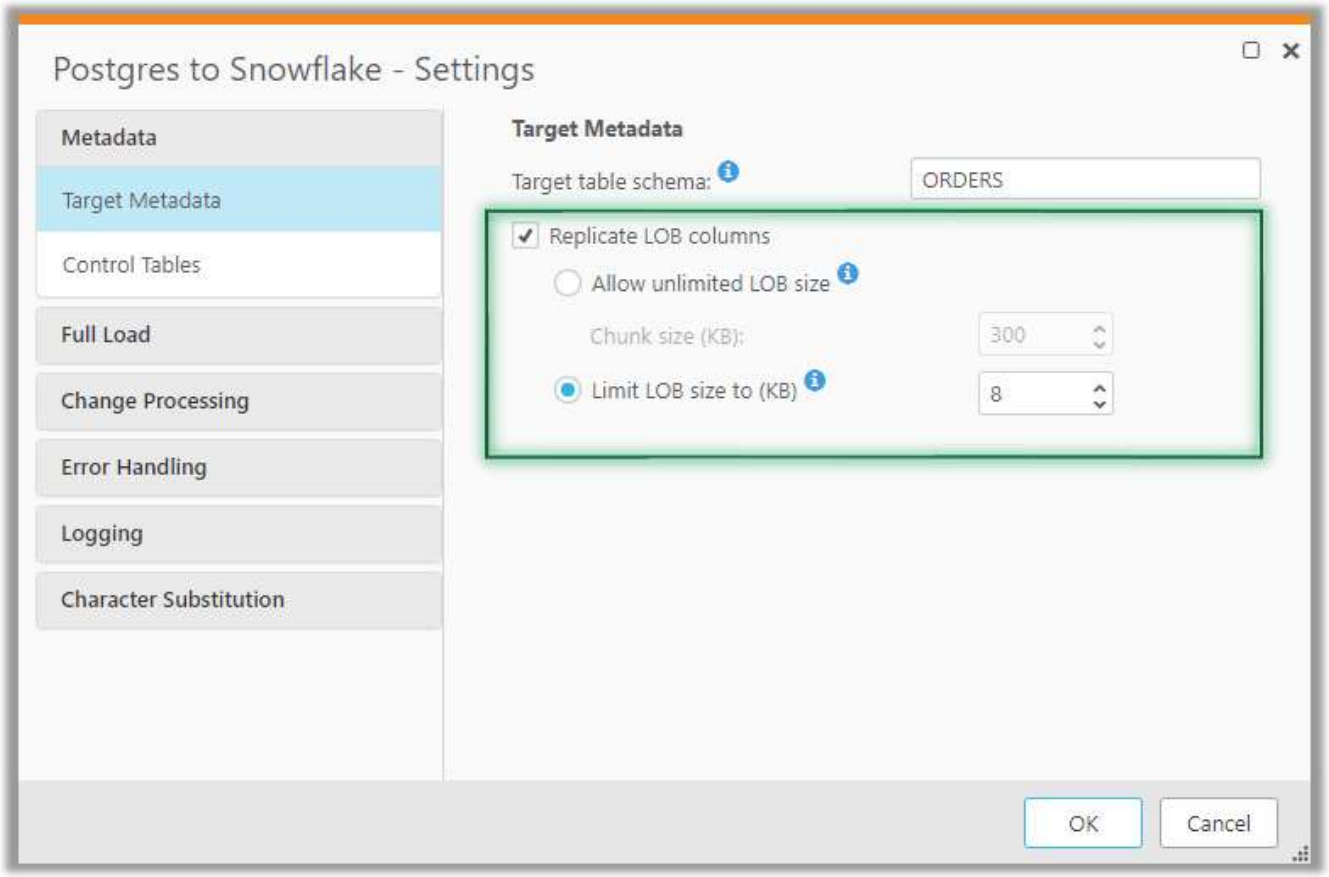


Figure 5 – LOB column optimizations.

It is recommended to set this option when transferring JSON documents based on the amount of data that the user wants to replicate to the target table. However, data truncation may occur if the specified size is less than the size of data stored in the source LOB column. For additional information on how to limit the size of a LOB column and other options, please refer to the Qlik Replicate documentation.

Finally, Qlik Replicate's automatic data-type conversion is dependent upon the data types supported by the source and target endpoints. Please review the Qlik Replicate documentation for a complete list of supported data types.

Snowflake Support for LOB and NCLOB Data Types

In Snowflake, LOB columns such as NCLOB columns are not supported, but Qlik Replicate can map LOB columns to other Snowflake data types such as VARCHAR; thus, JSON documents can be stored in Snowflake if VARCHAR data type is used.

Qlik Compose for Data Warehouses

Adding New Attributes to the Data Model

Qlik Compose for Data Warehouses automates the creation, code-generation, and change-data-capture strategy of a data warehouse. Users must complete at least three main functions when implementing Qlik Compose:

1. Defining a set of database connections
2. Creating a warehouse data model
3. Making a collection of ETL mappings that maintain the data warehouse

A fourth step is also required if users want to build data marts and star-schemas to support their analytics and business intelligence processes.

Qlik Compose allows users to create data models and define relationships between tables (entities). When creating a data model, users can either import existing data models from third-party data-modeling tools such as Erwin, or manually build their own data models. The data modeling feature allows users to discover table-relationships already defined in a source system (i.e. import existing table metadata) – if the source database is supported. Users can also add new attributes (columns) to the entities in the data model and create complex expressions to populate these new attributes. Lastly, these attributes can be defined as Type 1 or Type 2 attributes to maintain current or historical data changes respectively, in the data warehouse.

Figure 6 shows the logical view of a Qlik Compose entity called SRC_CUSTOMER that was added to the data model by importing the table structure from Snowflake. Incidentally, this table was created and populated by Qlik Replicate from the previous section of this whitepaper.

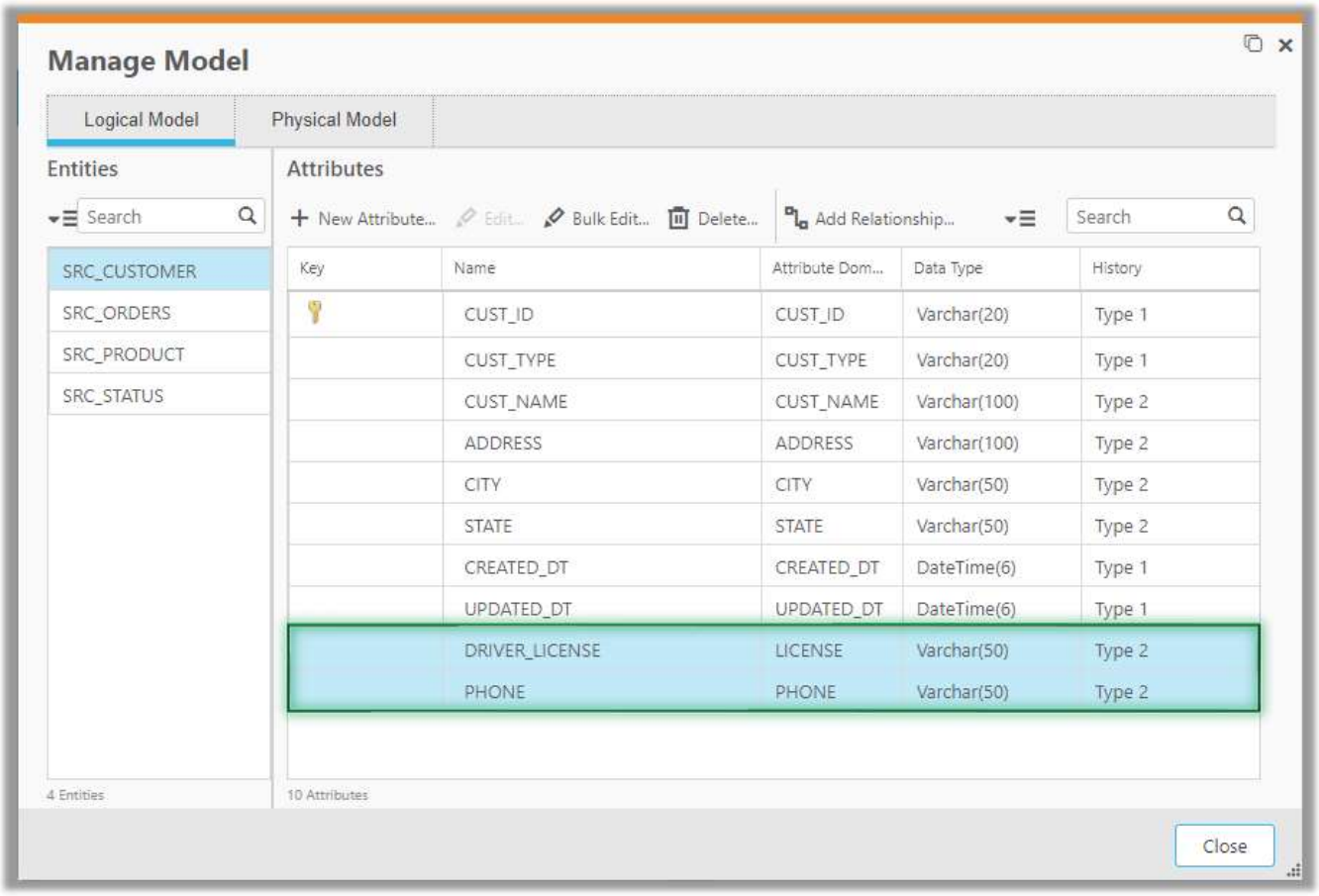


Figure 6 - A Qlik Compose entity with new attributes.

This entity SRC_CUSTOMER in Figure 6 has been modified as follows:

- The CUST_CREDIT_REPORT column has been removed from this entity because the user is interested on a sub-set of attributes from the JSON document. Therefore, it's not necessary to store the large attribute in the data warehouse.
- Two new attributes have been added to this entity: DRIVER_LICENSE and PHONE. The user is interested on extracting and transforming the driver license number and the phone number from the JSON document. Also, notice that these two attributes have been configured as Type 2 attributes because the user is interested in keeping the attribute history in the data warehouse.
- CUST_NAME, ADDRESS, CITY and STATE have also been changed to Type 2 attributes as well.

Qlik Compose users have several options to populate the data for the new attributes:

- Build expressions in the data model.

- Build expressions in the warehouse and generate the EL-T instructions to populate and maintain these attributes.
- Build expressions in the data mart to populate and maintain these values in a star schema.

The next section describes how to use Qlik Compose to build expressions in the data warehouse to extract, load, and transform these new attributes using Qlik Compose mappings.

Building Expressions to Extract, Load, and Transform Semi-Structured Data

One of the best features of Qlik Compose is the ability to auto-generation mappings and EL-T instructions that create and maintain a data warehouse. Mappings and EL-T instructions are generated based on the data model which are then managed in the data warehouse.

Figure 7, below, illustrates the Qlik Compose user interface. Users define EL-T instructions that transform data via Qlik Compose expressions which themselves are defined in the Model, Data Warehouse, or Data Mart sections.

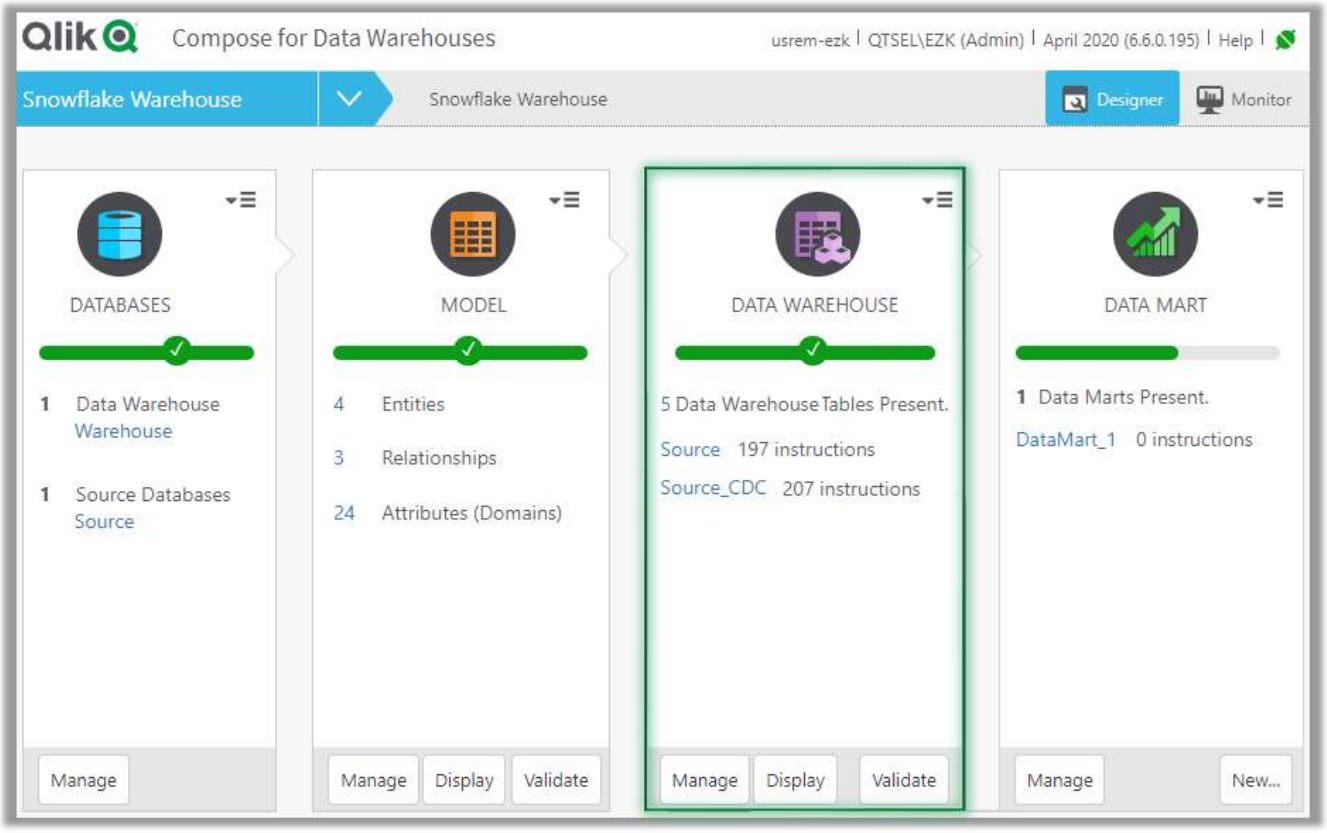


Figure 7 - Qlik Compose for Data Warehouses UI.

Note. Expressions defined in the Data Warehouse section must be defined inside of a mapping. Also, Qlik Compose expressions can utilize the built-in functions of the underlying data warehouse database to build complex data transformations. These Qlik Compose expressions are then translated into EL-T instructions and executed against the warehouse database engine.

Figure 8, below, shows a mapping for a warehouse table called SRC_CUSTOMER.

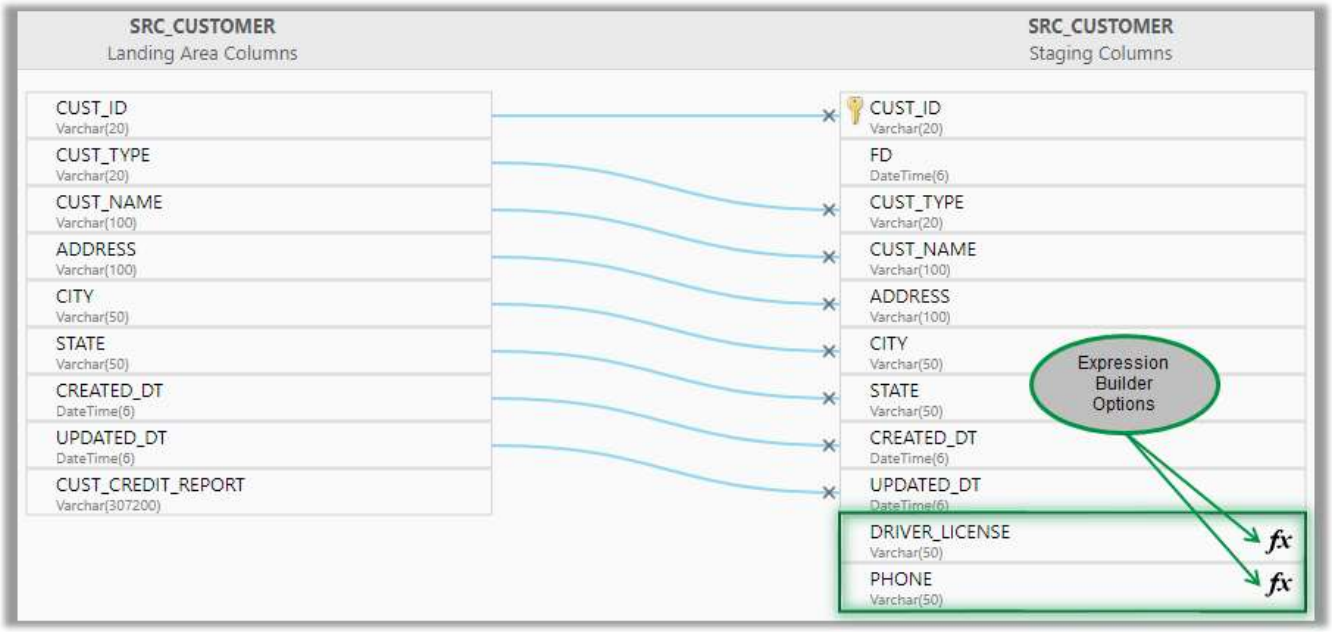


Figure 8 - Qlik Compose mapping with unmapped attributes

Most of the attributes featured in Figure 8 have been auto-mapped to a staging table that Qlik Compose uses to manipulate and transform the data before the warehouse table is populated. Notice that the newly added attributes, DRIVER_LICENSE and PHONE, have not been mapped to any of the landing attributes. We will manually map these attributes with a Qlik Compose expression. Figure 8 shows how clicking *fx* invokes the Expression Builder.

Figure 9 illustrates how to define a Qlik Compose expression that uses the PARSE_JSON function to extract the driver license number from the CUST_CREDIT_REPORT column.

Expressions Using Native Data Warehouse Functions

Qlik Compose users can take advantage of the built-in functions provided by a data warehouse to build sophisticated transformation expressions.

For instance, Snowflake exposes a PARSE_JSON function that extracts values from a table column that contains JSON data.

Figure 9 - An expression that extracts the driver license number.

Likewise, Figure 10 illustrates a second Qlik Compose expression to extract the primary applicant phone number. In this example, the phone numbers are originally stored in the JSON document with dots and the user wants to replace them with dashes. Therefore, the Qlik Compose expression includes an additional REPLACE function. The transformation expression now parses the phone number and replaces the dots (.) with a dashes (-).

Figure 10 - Qlik Compose expression to extract and format the phone number.

Qlik Compose expressions can be parsed and tested before generating the EL-T instructions. For additional information on how to parse and test expressions, please refer to the Qlik Compose documentation.

Once the expressions are tested, users can save the changes, and review the mapping. Figure 11 shows the mapping for the SRC_CUSTOMER after building the expressions for the staging attributes: DRIVER_LICENSE, and PHONE. Notice these new attributes are now mapped.

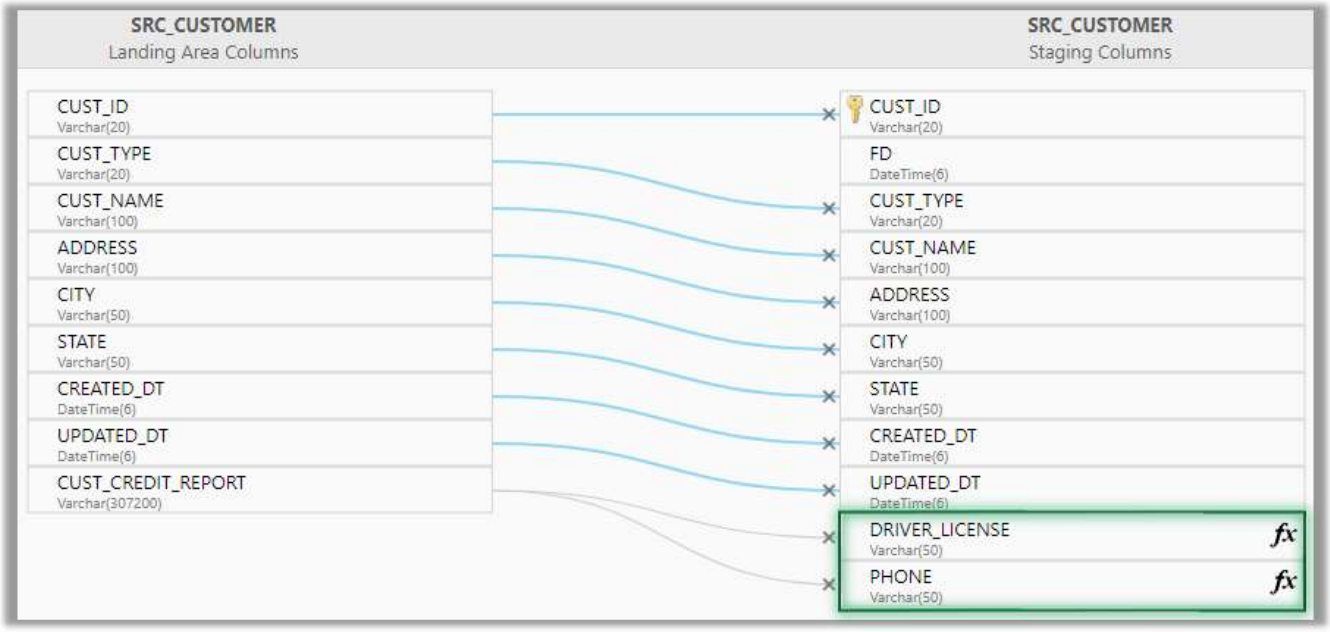
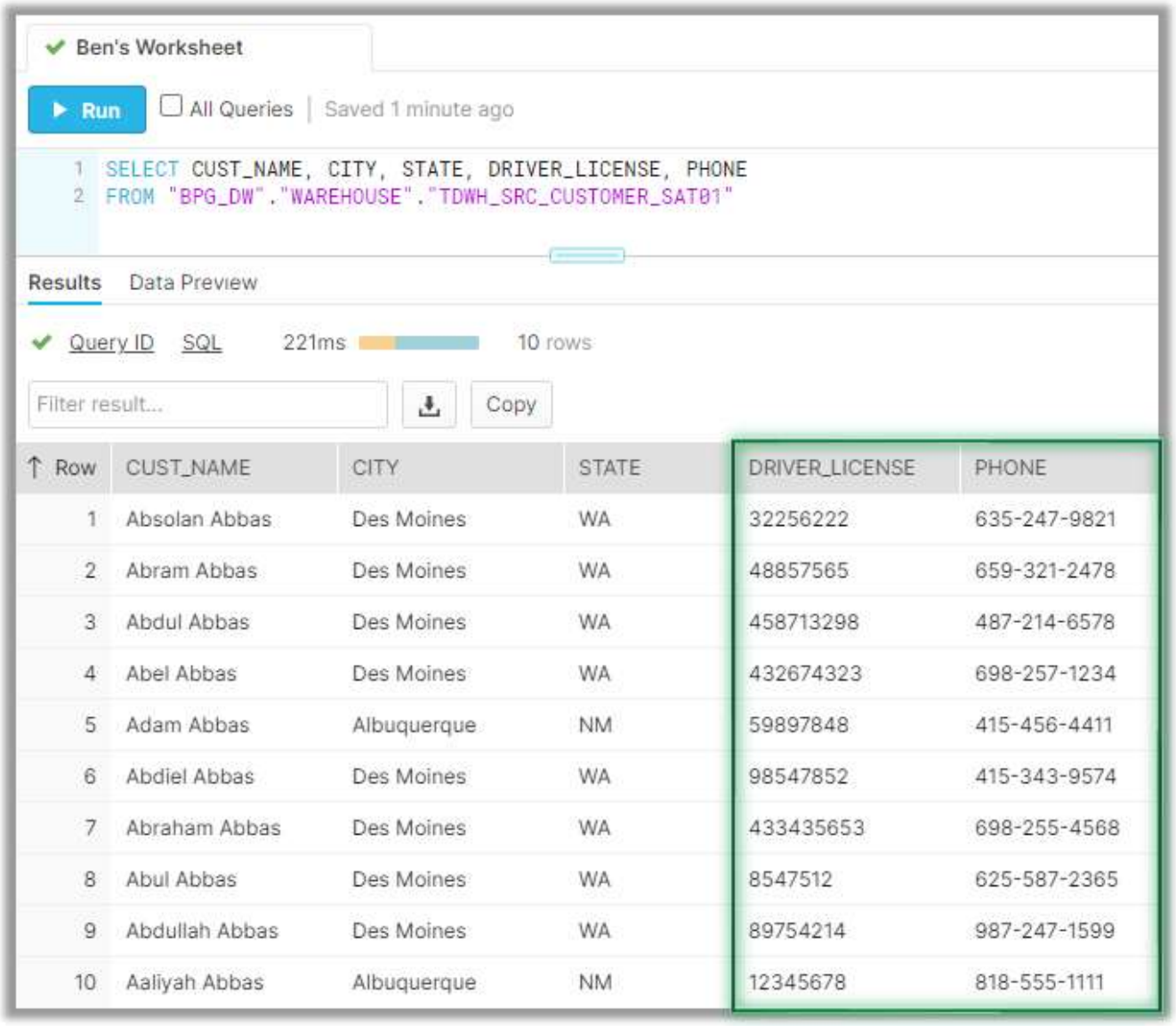


Figure 11 – Qlik Compose mapping with all attribute mapped.

The final step before populating the data warehouse table is the generation of the EL-T instructions. The EL-T instructions are based on the mapped attributes, including the JSON transformations defined for the DRIVER_LICENSE and the PHONE attributes. Once the EL-T instructions are generated successfully, the last step is to execute the EL-T instructions and perform the initial upload operation of this Qlik Compose warehouse table.



Row	CUST_NAME	CITY	STATE	DRIVER_LICENSE	PHONE
1	Absolan Abbas	Des Moines	WA	32256222	635-247-9821
2	Abram Abbas	Des Moines	WA	48857565	659-321-2478
3	Abdul Abbas	Des Moines	WA	458713298	487-214-6578
4	Abel Abbas	Des Moines	WA	432674323	698-257-1234
5	Adam Abbas	Albuquerque	NM	59897848	415-456-4411
6	Abdiel Abbas	Des Moines	WA	98547852	415-343-9574
7	Abraham Abbas	Des Moines	WA	433435653	698-255-4568
8	Abul Abbas	Des Moines	WA	8547512	625-587-2365
9	Abdullah Abbas	Des Moines	WA	89754214	987-247-1599
10	Aaliyah Abbas	Albuquerque	NM	12345678	818-555-1111

Figure 12 – Sample Snowflake table with transformed data.

Figure 12 show sample data from the warehouse table called TDWH_SRC_CUSTOMER_SAT01 in Snowflake that was created and populated by Qlik Compose. Notice the following two columns: DRIVER_LICENSE, and PHONE. The values for these columns were populated using the Qlik Compose expressions, built in the mapping. The DRIVER_LICENSE and the PHONE values are stored as clear text however the user can create additional Compose expressions to obfuscate these values using other built-in functions if required.

Conclusion

This article demonstrates how to replicate and transform semi-structured data using Qlik Replicate and Qlik Compose – specifically - the article discusses on how to replicate and transform JavaScript Object Notation (JSON) data – a popular semi-structured data format.

The steps described in this article can also be applied to other semi-structured data formats such as Extensible Markup Language (XML).

About Qlik

Qlik's vision is a data-literate world, one where everyone can use data to improve decision-making and solve their most challenging problems. Only Qlik offers end-to-end, real-time data integration and analytics solutions that help organizations access and transform all their data into value. Qlik helps companies lead with data to see more deeply into customer behavior, reinvent business processes, discover new revenue streams, and balance risk and reward. Qlik does business in more than 100 countries and serves over 50,000 customers around the world.

qlik.com