..

Advanced Analytics

Date published: 2020-10-30
Date modified: 2022-09-21

https://docs.cloudera.com/

Legal Notice

Please visit the Cloudera software product page for more information on Cloudera software. For more information on Cloudera support services, please visit either the Support or Sales page. Feel free to contact us directly to discuss your specific needs.

Cloudera reserves the right to change any products at any time, and without notice. Cloudera assumes no responsibility nor liability arising from the use of products, except as expressly agreed to in writing by Cloudera.

Disclaimer: EXCEPT AS EXPRESSLY PROVIDED IN A WRITTEN AGREEMENT WITH CLOUDERA, CLOUDERA DOES NOT MAKE NOR GIVE ANY REPRESENTATION, WARRANTY, NOR COVENANT OF ANY KIND, WHETHER EXPRESS OR IMPLIED, IN CONNECTION WITH CLOUDERA TECHNOLOGY OR RELATED SUPPORT PROVIDED IN CONNECTION THEREWITH. CLOUDERA DOES NOT WARRANT THAT CLOUDERA PRODUCTS NOR SOFTWARE WILL OPERATE UNINTERRUPTED NOR THAT IT WILL BE FREE FROM DEFECTS NOR ERRORS, THAT IT WILL PROTECT YOUR DATA FROM LOSS, CORRUPTION NOR UNAVAILABILITY, NOR THAT IT WILL MEET ALL OF CUSTOMER’S BUSINESS REQUIREMENTS. WITHOUT LIMITING THE FOREGOING, AND TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW, CLOUDERA EXPRESSLY DISCLAIMS ANY AND ALL IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY, QUALITY, NON-INFRINGEMENT, TITLE, AND FITNESS FOR A PARTICULAR PURPOSE AND ANY REPRESENTATION, WARRANTY, OR COVENANT BASED ON COURSE OF DEALING OR USAGE IN TRADE.

Contents

Advanced Analytics overview

Cloudera Data Visualization provides an extensive array of tools to support advanced analysis.

Complex data types

Cloudera Data Visualization supports native complex data type configurations, bypassing ETL workflows and avoiding unnecessary flattening at query time. Explore complex data types to leverage the powerful native SQL already optimized for complex types.

Dimension hierarchies

Cloudera Data Visualization provides full support for dimensional hierarchy modelling at dataset level, enabling smooth natural transition between granularity levels of data. For more information, see Dimension hierarchies.

Segments

Segments can be very powerful in data visualization, and form the foundation of cohort analysis and row-level access restrictions. For more information, see Segments.

Events

Identifying and managing events within your time-stamped data enables you to measure outcomes of complex workflow scenarios. For more information, see Events.

Visuals for analytics

For advanced analysis, you can create advanced visual types, such as funnels, flows, correlation flows, and histograms. For more information, see Visuals for Analytics.

Advanced visualization techniques

Advanced visualization techniques, such as trellises, dual-axis line visuals, and dimension hierarchies free you to explore the visualized data from diverse perspectives. For more information, see Advanced Visualization Techniques.

Analytic functions

On select connection types, Cloudera Data Visualization supports several analytic functions that examine overlapping groupings of data. For more information, see Supported connections.

Derived data

You can derive data from an existing query, build computed columns for re-use, and reference results in new queries. Derived data enables you to stack sub-query analytical results.

Related Information

Complex data types
Dimension hierarchies
Segments
Events
Visuals for analytics
Advanced visualization techniques
Derived data
Supported Connections

Complex data types

Cloudera Data Visualization understands native complex data type configurations, and automatically translates complex structures into intuitive drag and drop dataset field configurations.

Native support for complex datatypes allows us to bypass ETL workflows and avoid unnecessary flattening at query time. It also leverages the powerful native SQL already optimized for complex types.

Note:

- Impala connections support the STRUCT, ARRAY, and MAP complex data types.
- For Hive connections, support is limited to the STRUCT data type.

Note:

When describing how Data Visualization supports complex data types, data and structures are adapted from the TPC-H benchmark as a Sample schema and data for experimenting with Impala complex types.

Overview of complex data types

- *STRUCT*

  A group of named fields, where each field can have a different data type, both primitive and complex.

  See STRUCT data type.
- *ARRAY*

  A list of values that share the same data type.

  See ARRAY data type.
- *MAP*

  A set of (key,value) pairs.

  See MAP data type.

The elements of an ARRAY or a MAP, or the fields of a STRUCT may be primitive, or they may be other complex types. You can construct elaborate data structures with many levels, such as a ARRAY with STRUCT elements.

Appearance of complex data types

Cloudera Data Visualization presents complex types very simply. Users can manipulate the individual component fields of complex types in the same manner as the primitive data type fields: place them on the shelves of the visual, as a component of an expressions, as a filter, and so on. Each level of a complex data type may be expanded to show component details, or collapsed.

Cloudera Data Visualization processes complex types as Dimensions, grouping individual elements under the complex type. The components of a complex type cannot be re-assigned to the Measures group even if they logically represent a measurement; however, you can easily use them as measurements on visual shelves, in filters, and so on.

Because native SQL processing is supported, you do not have to worry about the complexities of query generation to access these elements.

Restrictions

Complex data types have the following implementation restrictions:

- MAP, ARRAY, and STRUCT fields require base tables or partitions that use the Parquet file format.
- You cannot use the fields that contain complex data types as partition keys in a partitioned table.
- The Compute Statistics operation does not work for complex data types.
- The column definition for any complex type, including nested types, can have a maximum of 4000 characters.

- The Import into Dataset feature does not support files that contain complex datatypes.
- Analytical Views do not support complex type queries, because complex types require JOIN operations. As a result, visuals that use complex data types do not have valid recommendations. However, if you associate a logical view with the dataset, then the recommendation engine proceeds to generate suggestions, and presents the relevant dataset columns as complex data types.

Related Information

TPC-H

Sample Schema and Data for Experimenting with Impala Complex Types

STRUCT data type

ARRAY data type

MAP data type

STRUCT data type

This article describes the specifics of the STRUCT complex data type.

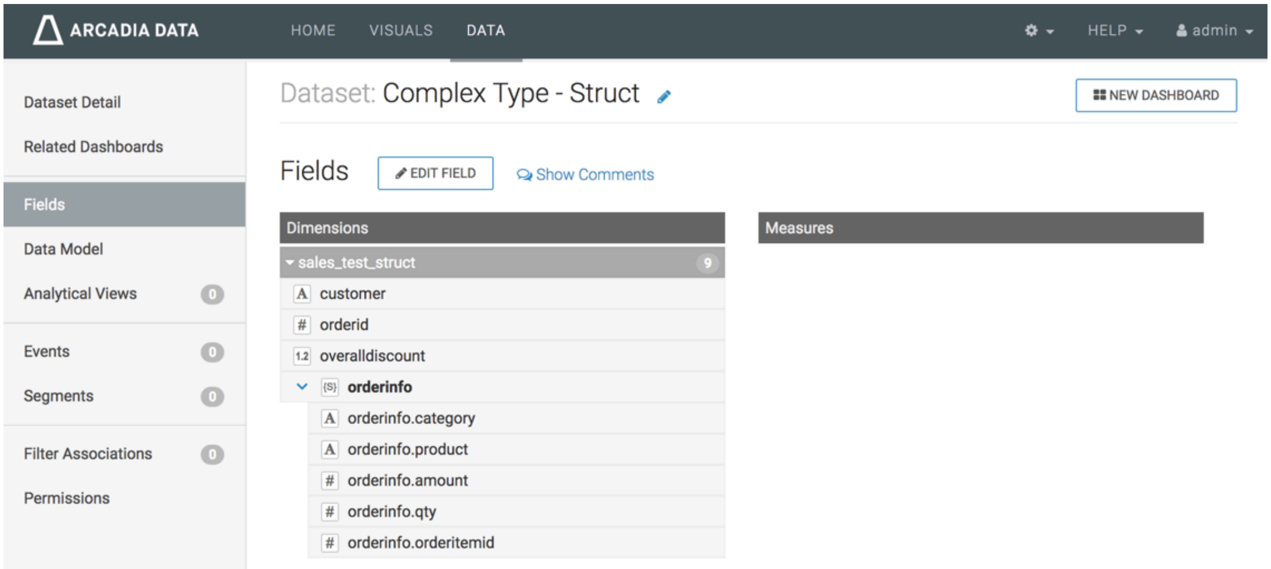
Syntax for STRUCT

```
column_name STRUCT < name : type [COMMENT 'comment_string'], ... >
type ::= primitive_type | complex_type
```

A STRUCT has a fixed number of named fields, and each field can be a different type. A field within a STRUCT can also be another STRUCT, or an ARRAY or a MAP

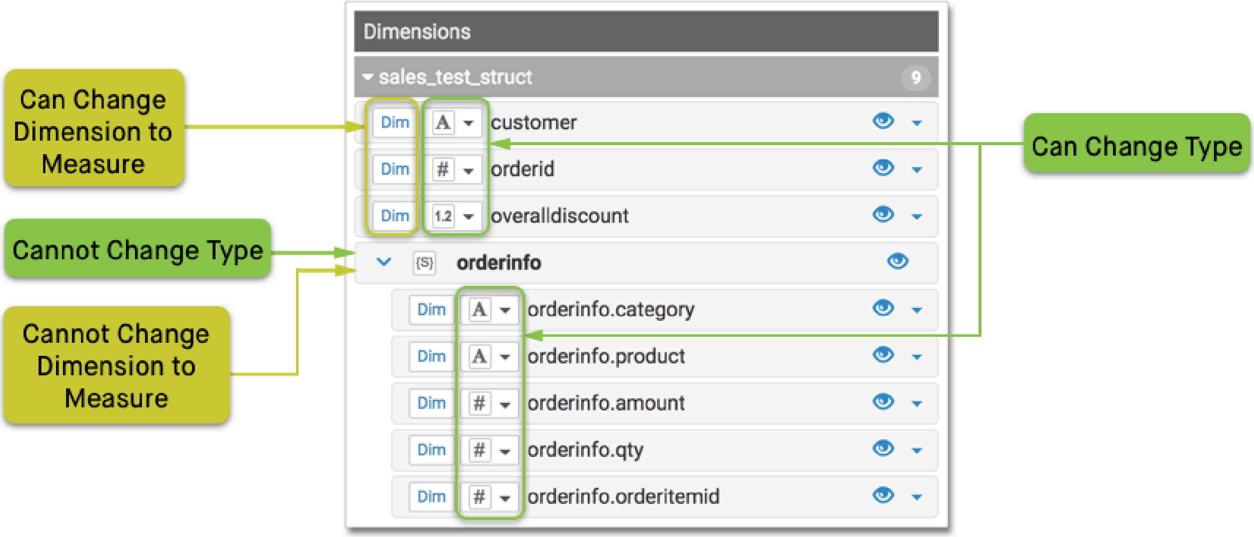
STRUCTs in the Dataset Field Interface

In the Dataset Fields interface, an example of a basic STRUCT data type may look something like the following image. Notice that each level of a complex data type may be expanded to show component details, or collapsed.



In the example of the dataset Complex Type - Struct, you can see that the Dimensions Customer (String, with the symbol A), Orderid (an Integer, symbolized by #), and overalldiscount (a Real, symbolized by 1.2) are primitive types. However, the Dimensions orderinfo is a Struct data type, symbolized by [S].

When we click Edit Fields, we can see that while primitive types can be cast as alternate data types (such as Integer into Real), the complex data type STRUCT cannot be changed to another type. However, the primitive components of the array may be cast as other primitive data types. Additionally, unlike other data types, uses complex datatypes only as Dimensions; they or their components cannot be re-defined as Measurements of the dataset.



Can Change Dimension to Measure

Cannot Change Type

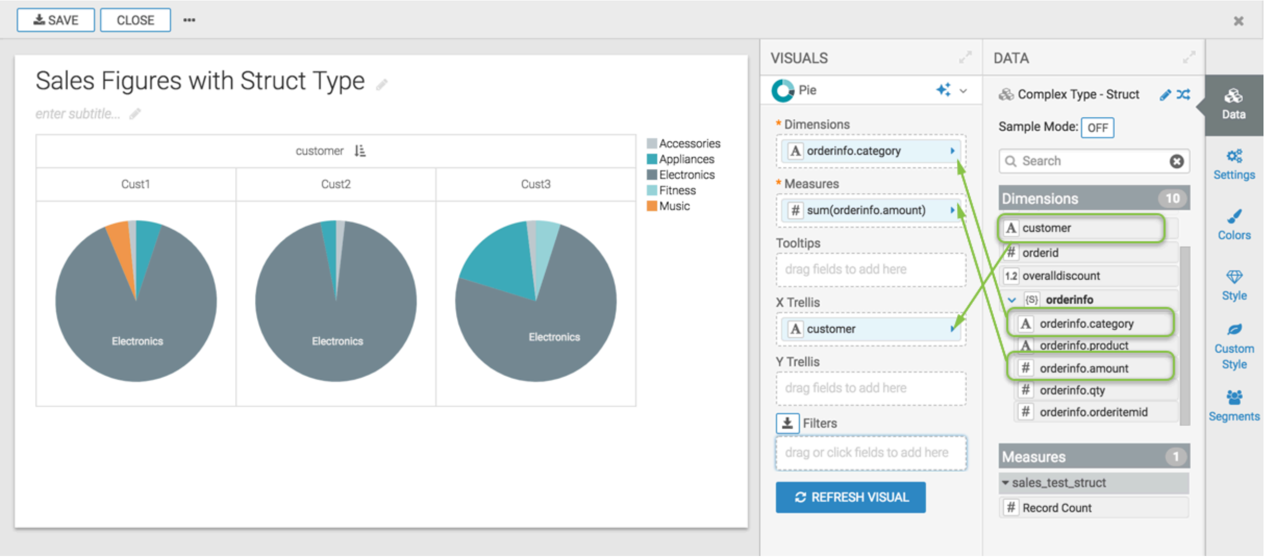
Cannot Change Dimension to Measure

Can Change Type

STRUCTs in Visuals

When building a visual with complex data, you cannot use the complex type directly, as a whole. However, you can add the primitive components of the complex type to the shelves of the visual.

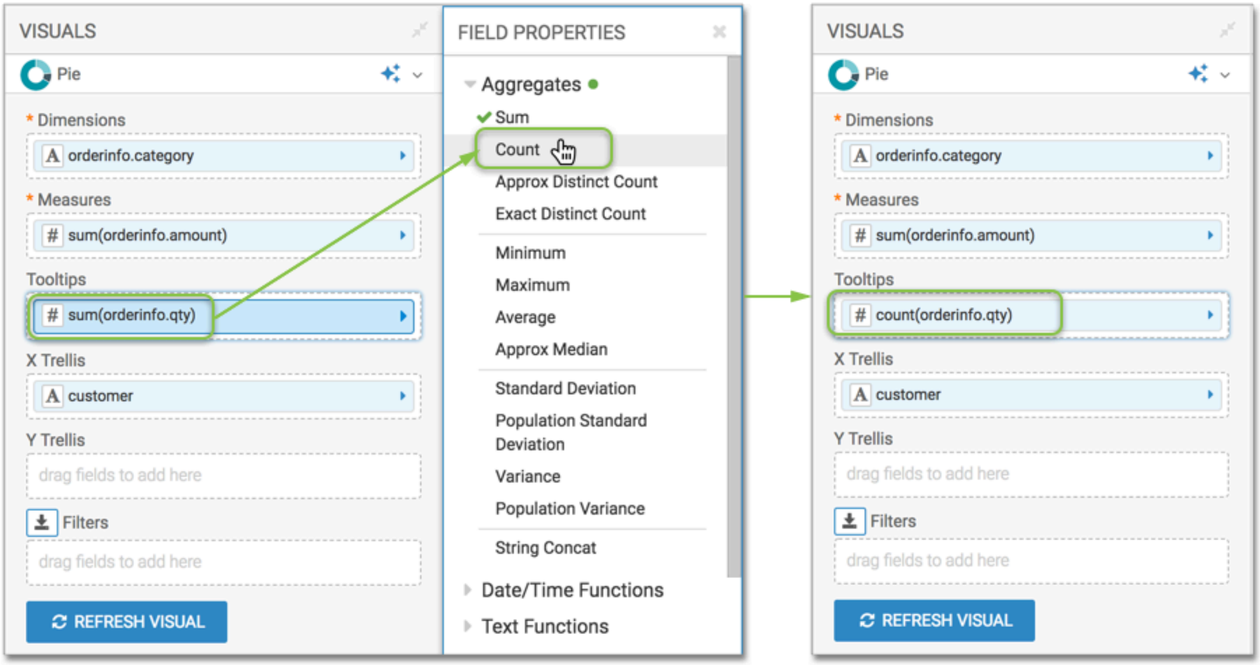
In the following illustration, we built a Pie visual with Customer Name on the X Trellis shelf, the orderinfo.category component on the Dimensions shelf, and the orderinfo.amount component on the Measures shelf.



Changing Field Properties

It is very simple to change field properties for a component of a complex data type.

In the following illustration, we change the orderinfor.qty component on the Tooltips shelf from the default max() aggregation to the count() function.



STRUCT in Expression Editor

The expression editor supports the full use of STRUCTs, both in Dataset and Visual interfaces.

ARRAY data type

This article describes the specifics of the ARRAY complex data type.

Syntax for ARRAY

```
column_name ARRAY < type >
type ::= primitive_type | complex_type
```

ARRAY data types represent collections with arbitrary numbers of elements, where each element is the same type. An ARRAY type is like a miniature table; the table equivalent has two columns:

- *POS*

  Position of element in the array.

  Access as array_name.pos.
- *ITEM*

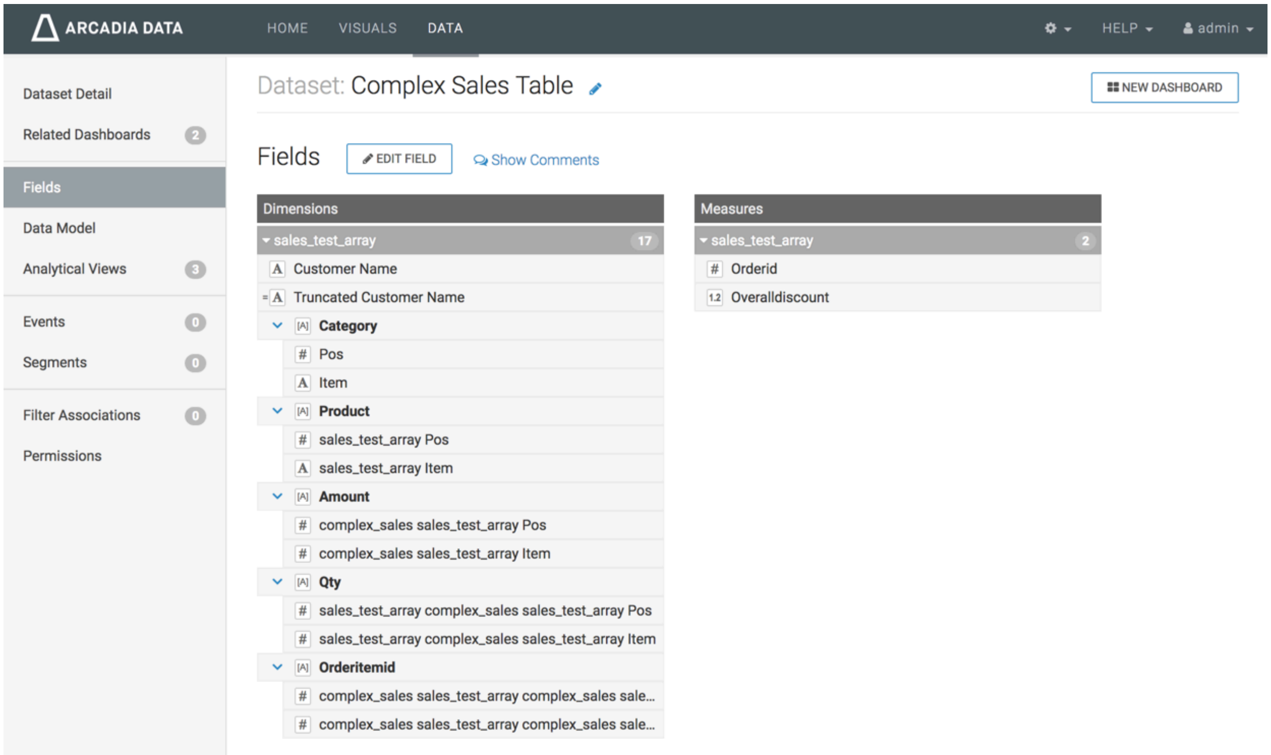
  Value of array element. May be a scalar, or another complex type (another ARRAY, a STRUCT, or a MAP)

  Access as array_name.item.

  If an array contains a STRUCT, access as array_name.item.field_name, or as array_name.field_name.

Arrays in the Dataset Field Interface

In the Dataset Fields interface, an example of a basic ARRAY data type may look something like the following image. Notice that each level of a complex data type may be expanded to show component details, or collapsed.



Dataset: Complex Sales Table

Fields

Dimensions

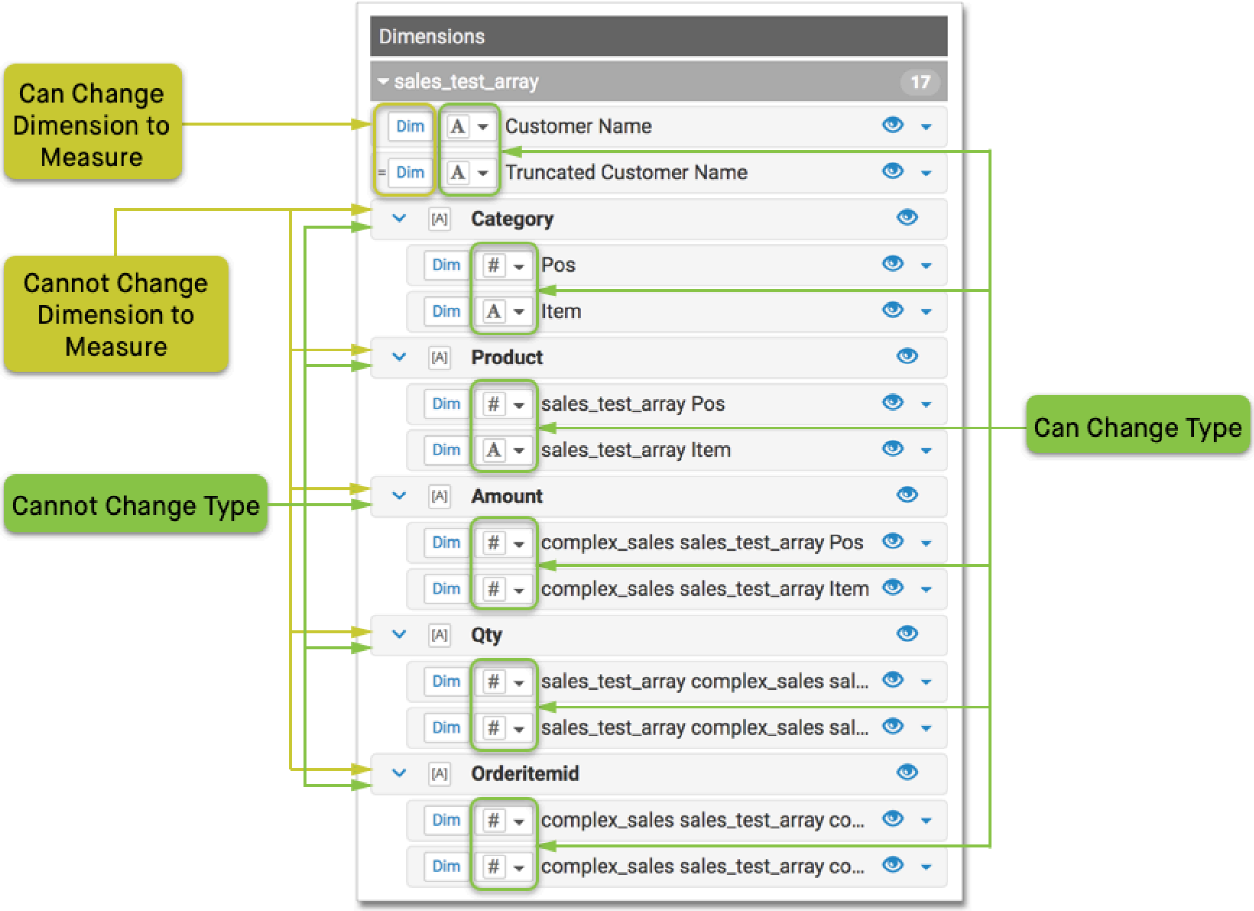
- sales_test_array 17
  - A Customer Name
  - A Truncated Customer Name
  - [A] Category
    - # Pos
    - A Item
  - [A] Product
    - # sales_test_array Pos
    - A sales_test_array Item
  - [A] Amount
    - # complex_sales sales_test_array Pos
    - # complex_sales sales_test_array Item
  - [A] Qty
    - # sales_test_array complex_sales sales_test_array Pos
    - # sales_test_array complex_sales sales_test_array Item
  - [A] Orderitemid
    - # complex_sales sales_test_array complex_sales sale...
    - # complex_sales sales_test_array complex_sales sale...

Measures

- sales_test_array 2
  - # Orderid
  - 1.2 Overalldiscount

In the example of the dataset Complex Sales Table, you can see that the Dimensions Customer Name and the derived Truncated Customer Name are primitive types (both are Strings, marked with the symbol A), along with the Measures Orderid (an Integer, symbolized by #) and Overalldiscount (a Real, symbolized by 1.2). However, the Dimensions Category, Product, Amount, Qty and Orderitemid are all Array data types, symbolized by [A].

When we click Edit Fields, we can see that while primitive types can be cast as alternate data types (such as Integer into Real), the complex data type Array cannot be changed to another type. However, the primitive components of the array may be cast as other primitive data types. Additionally, unlike other data types, uses complex datatypes only as Dimensions; they or their components cannot be re-defined as Measurements of the dataset.



Can Change Dimension to Measure

Cannot Change Dimension to Measure

Cannot Change Type

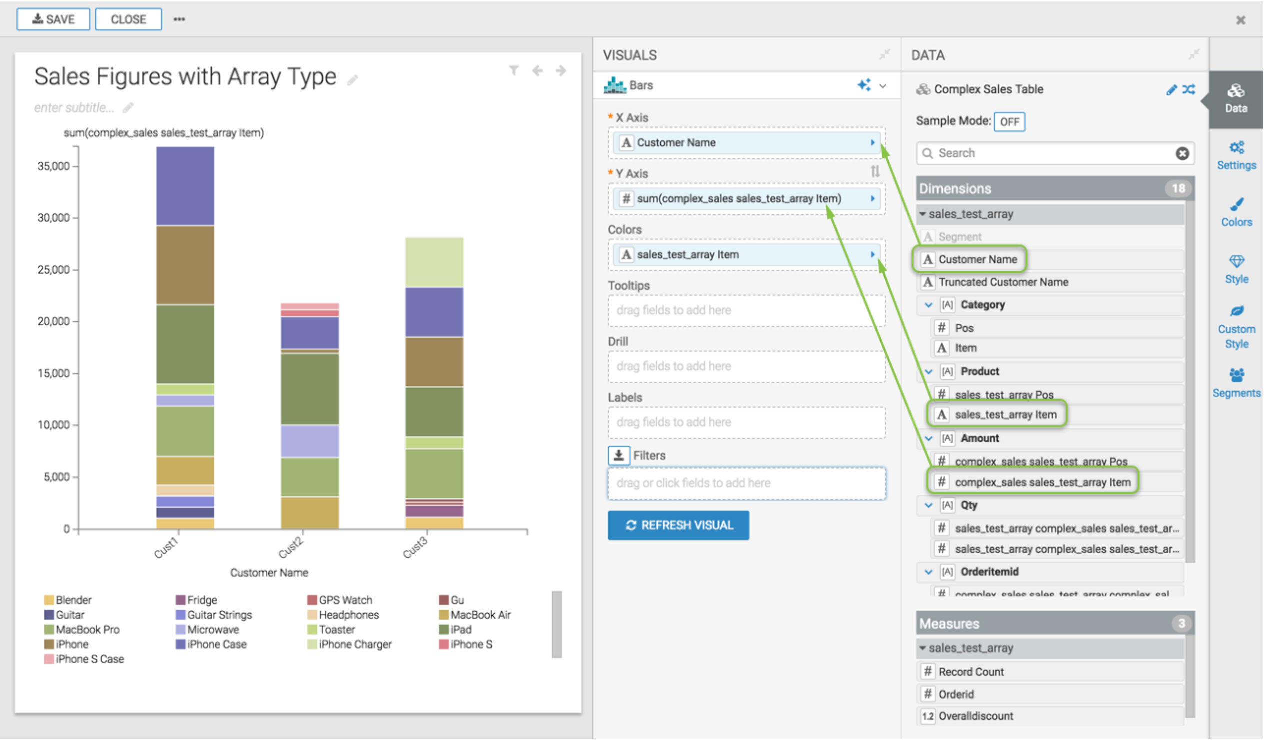
Can Change Type

Dimensions
sales_test_array
17
Dim A Customer Name
Dim A Truncated Customer Name
[A] Category
Dim # Pos
Dim A Item
[A] Product
Dim # sales_test_array Pos
Dim A sales_test_array Item
[A] Amount
Dim # complex_sales sales_test_array Pos
Dim # complex_sales sales_test_array Item
[A] Qty
Dim # sales_test_array complex_sales sal...
Dim # sales_test_array complex_sales sal...
[A] Orderitemid
Dim # complex_sales sales_test_array co...
Dim # complex_sales sales_test_array co...

Arrays in Visuals

When building a visual with complex data, you cannot use the complex type directly, as a whole. However, you can add the primitive components of the complex type to the shelves of the visual.

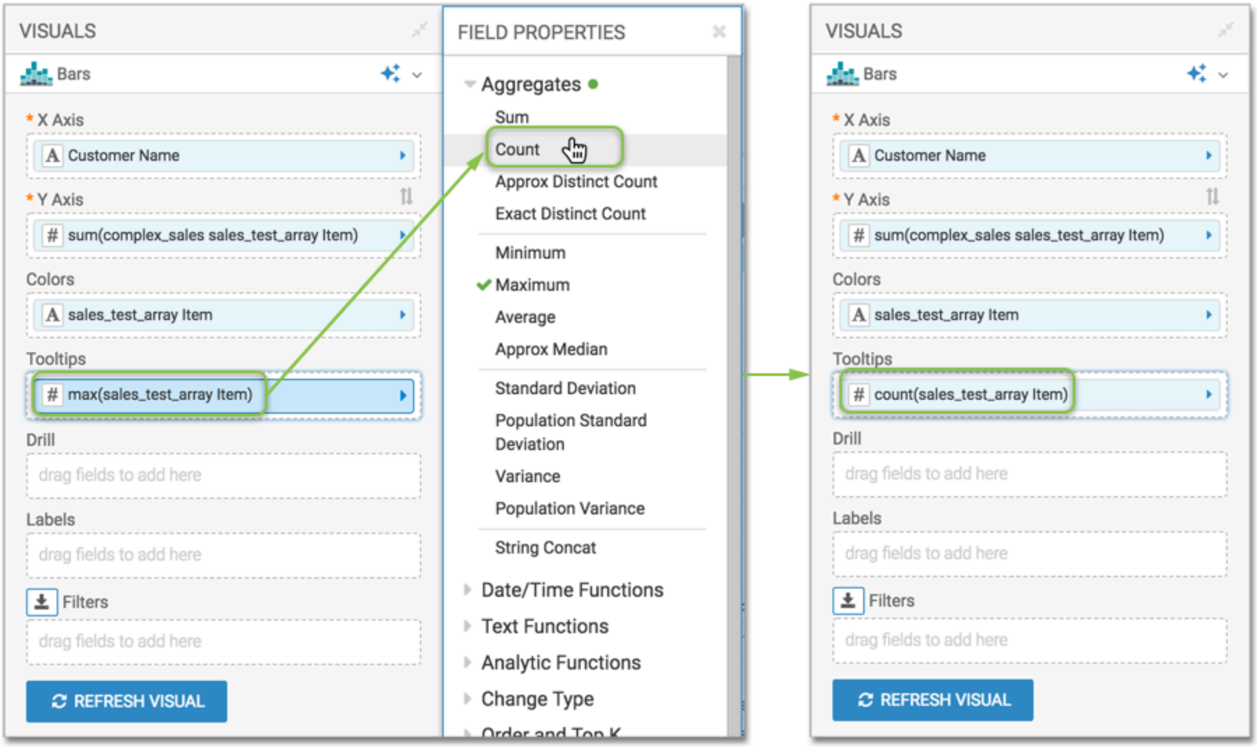
In the following illustration, we built a Bars visual with Customer Name on the X Axis shelf, the Amount:Item component on the Y Axis shelf, and grouped on the Colors shelf by Product:Item component.



Changing Field Properties

It is very simple to change field properties for a component of a complex data type.

In the following illustration, we change the Product:Item component on the Tooltips shelf from the default max() aggregation to the count() function.



Arrays in Expression Editor

The expression editor supports the full use of Arrays, both in Dataset and Visual interfaces.

MAP data type

This article describes the specifics of the MAP complex data type.

Syntax for MAP

```
column_name MAP < primitive_type, type >
type ::= primitive_type | complex_type
```

MAP data types represent sets of key-value pairs. These collections have an arbitrary number of elements, where each element is the same type. A MAP type is like a miniature table; the table equivalent of a scalar (primitive) values that has two columns:

- *KEY*

  A scalar type, such as BIGINT, STRING, TIMESTAMP, which enables you to define non-continuous sequences or categories with arbitrary names. The keys in a map column may represent a numeric sequence of events during a manufacturing process, or TIMESTAMP values that correspond to sensor observations.

  Access as map_name.key.
- *VALUE*

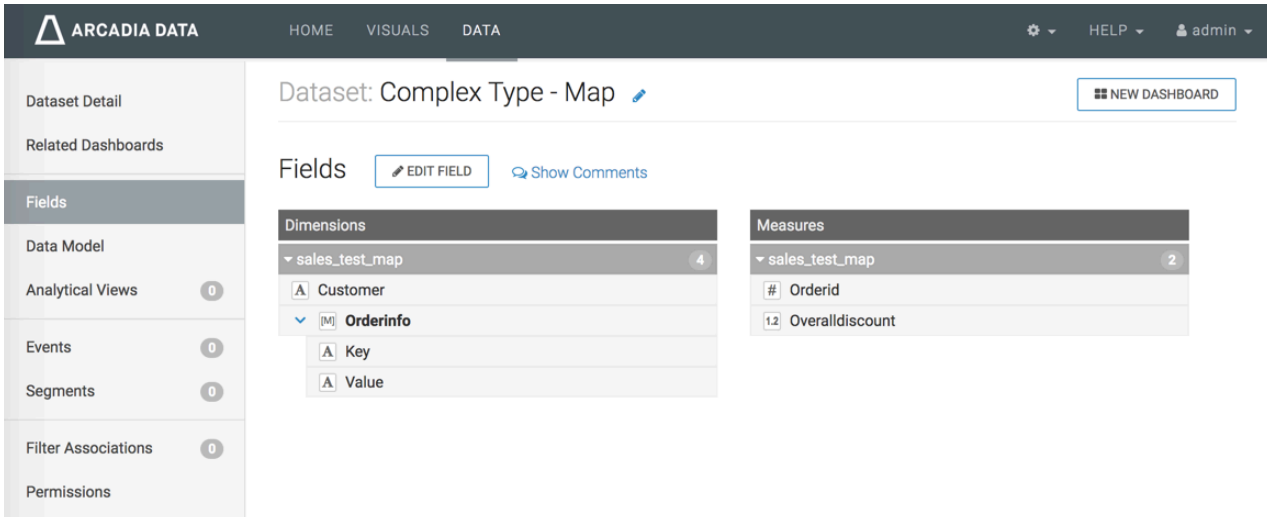
  The second part of a key-value pair in a map. May be a scalar, or another complex type (an ARRAY, a STRUCT, or another MAP).

  Access as map_name.value.

  If a map contains a STRUCT, access as map_name.value.field_name, or as map_name.field_name.

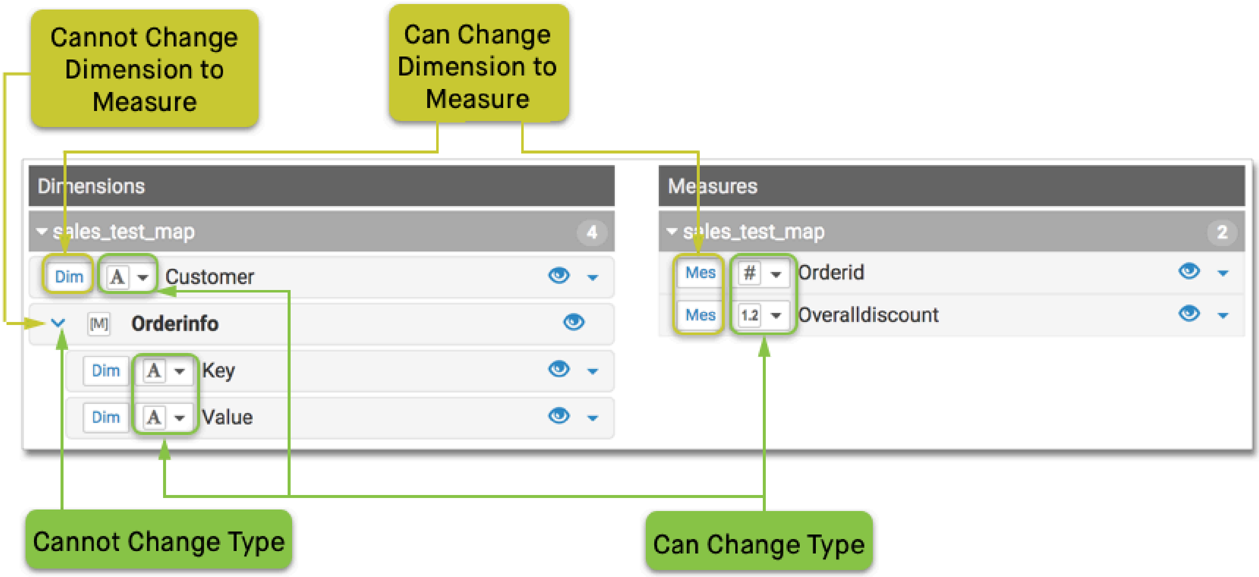
MAPs in the Dataset Field Interface

In the Dataset Fields interface, an example of a basic MAP data type may look something like the following image. Notice that each level of a complex data type may be expanded to show component details, or collapsed.



In the example of the dataset Complex Type - Map, you can see that the Dimension Customer (a String, symbolized by A), and Measures Orderid (an Integer, symbolized by #) and Overalldiscount (a Real, symbolized by 1.2) are primitive types. However, the Dimension Orderinfo is a Map data type, symbolized by [M].

When we click Edit Fields, we can see that while primitive types can be cast as alternate data types (such as Integer into Real), the complex data type Array cannot be changed to another type. However, the primitive components of the array may be cast as other primitive data types. Additionally, unlike other data types, uses complex datatypes only as Dimensions; they or their components cannot be re-defined as Measurements of the dataset.



MAPs in Visuals

When building a visual with complex data, you cannot use the complex type directly, as a whole. However, you can add the primitive components of the complex type to the shelves of the visual.

Changing Field Properties

It is very simple to change field properties for a component of a complex data type. You can change the aggregation function of a component, its type, the display options, and so on, just like a standalone primitive type field.

MAPs in Expression Editor

The expression editor supports the full use of MAPs, both in Dataset and Visual interfaces.

Dimension hierarchies

Cloudera Data Visualization provides full support for dimensional hierarchy modelling at dataset level, enabling smooth natural transition between granularity levels of data.

Hierarchies are a natural form of data organization. For example, you can break up any given time interval into intervals of shorter duration based on years, months, weeks, days, and so on. Similarly, the territory of United States consists of well-defined states, then counties, then municipalities and incorporated and unincorporated areas.

Hierarchies may only be defined through columns that are classified as dimensions. If necessary, change the column attribute from Measure to Dimension. Alternatively, clone a column, change it to a dimension, and only then use it in defining a dimensional hierarchy.

Just like with column names, the name of the dimensional hierarchy must be unique to the dataset.

Columns that define a dimension hierarchy are retained as a reference, not a copy. Therefore, changes to the basic definition of the column (such as name, type, calculation) propagate into the dimensional hierarchy.

To build out dimensional hierarchies, use 'drag and drop' action to add dimension columns, and to prioritize them. Cloudera recommends that you use the delete function to permanently remove items from a hierarchy. You can also move elements from one hierarchy to another; note that this action removes the dimension from the source hierarchy and adds it to the target hierarchy.

Segments

With Cloudera Data Visualization, you can create and manage segments within your data.

Segments can be used to analyze subsets of the data that have specified properties. Normally, segments are defined by filtering data based on dimensions or measures. Discovering trends within segments and making comparisons across segments provides powerful insights into the data and the business it represents.

To create and manage segments in the Dataset Segments user interface, see the following topics:

- Creating Segments
- Cloning Segments
- Editing Segments
- Creating Entity Segments
- Deleting Segments

You can also segments from filter definitions in a visual. See, Creating segments from filters

To view how segments work in a visual, see Using segments in visuals.

To see how you can use natural segments to restrict data access through filter association, see Setting segments .

Related Information

Creating segments

Cloning segments

Editing segments

Creating entity segments

Deleting segments

Creating segments from filters

Using segments in visuals

Setting segments

Events

With Cloudera Data Visualization, you can identify and manage events within your time-stamped data.

Defining events in a time-stamped dataset enables you to sequence them chronologically, and to define business scenarios that can be modelled by flow and funnel visuals. See Flows and Funnels.

This includes the following topics:

- Creating Events
- Cloning Events
- Editing Events
- Deleting Events

Related Information

Creating events

Cloning events

Editing events

Deleting events

Correlation flows

Visuals for analytics

Cloudera Data Visualization has a rich catalog of visual types to assist you in analyzing your data.

The following visual types may be of more interest for advanced analysis:

Related Information

Combo visuals

Gauges

Bullets

Box plots

Networks

Dendograms

Correlation flows

Histograms

Combined bar/line visuals

In CDP Data Visualization, Combined Bar/Line (Combo) visuals unite bar and line plots to enable easy comparison of trends and measures across the same dimensions.

Note: This visual type supports smooth rendering of streaming data.

The bars and the line can share the same vertical axis, or use a secondary axis. For more information, see Creating basic combo visuals and Assigning a measure to a secondary axis.

You can optionally render the line portion of the visual as an area. For more details, see Change the line to an area.

Related Information

Creating basic combos

Assigning a measure to a secondary axis

Change the line to an area

Advanced visualization techniques

Cloudera Data Visualization offers advanced analytical and visualization options that work with its applications.

The following examples illustrate some advanced analysis techniques:

Related Information

Trellised visuals

Analyzing multiple measures

Creating line visuals with dual axes

Working with dimension hierarchies

Working with hierarchical cross-tabulation visuals

Adding highlights

Highlighting elements by hover-over legend

Trellised visuals

Trellis charts are grouped sets of visuals that represent different partitions (segments) of the dataset in the same chart type, scale, and axes. This makes the comparison of sub-groups very intuitive. Trellis charts are sometimes called lattice charts, grid charts, panel charts, or small multiples.

Any chart type that uses the X and Y axes explicitly may be trellisable, depending on the dataset. What is required is an extra entry on either the X or Y shelf; the first field on the shelf represents the segment definition of the trellis, and the second field is the actual dimension. Some charts types have optional X and Y shelves so that trellises may be enabled.

A dataset may be double-trellisable — you could declare a 'partition' on both X and Y shelves at the same time.

There are two approaches for trellises:

- Commonly, we use Trellising on dimensions to show the sections of data in separate but related charts.
- We can also show different data aggregates, side by side, by placing extra measurements on the Measures shelf, to Trellis on measures.

You can find specific examples of trellised charts in the following articles:

- Radials
- Areas
- Word clouds

Some visual types (bars, lines, areas, and grouped bars) support trellising on measures.

Related Information

Trellising on dimensions

Trellis on measures

Radials

Areas

Word clouds

Derived data

Derived data lets you to reference results in new queries, query stacking, and eases cohort analysis. Cloudera Data Visualization uses derived data for computed fields in data modeling, weighted sums and averages, custom binning, for set-based and group-based analysis, and for combining data from different data sources.

Derived data enables you to reference query results in new queries, in essence "stacking" results from sub-select queries. The derived data feature also supports cohort analysis, where a set of data from a report is used (joined back) in another report, and allows you to build computed columns for re-use.

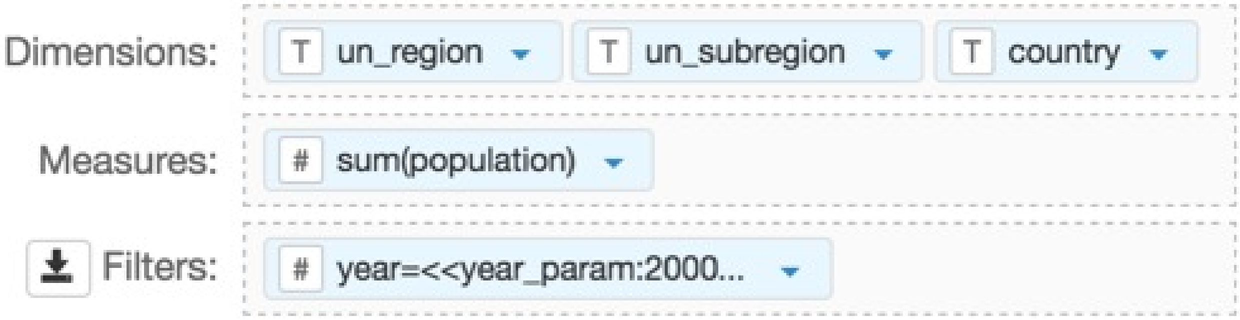
Note: By default, the Derived Data option is turned off. It can be turned on by a user with administrative privileges; see Enabling derived data.

Derived Data is very useful in determining weighted averages and other, more complex calculations. For example, in the dataset World Life Expectancy, life expectancy is reported at the level of each country, for each year. If you wanted to determine the life expectancy by region or subregion, you have to calculate a weighted average of life expectancies. You can also parametrize derived data definitions using bracket notation.

The following steps demonstrate how to use derived data on a table visual based on the dataset World Life Expectancy [data source samples.world_life_expectancy].

1. Place the fields un_region, un_subregion and country on the Dimension shelf.
2. Place the field population, as sum(population), on the Measures shelf.

3. Place the field year on the Filters shelf, and change the expression to [year]=<<year_param:2000>>. This enables you to dynamically change derived data calculations. You must specify a default value in the parametrized expression.



Dimensions: T un_region, T un_subregion, T country

Measures: # sum(population)

Filters: # year=<<year_param:2000...

Related Information

Defining derived data

Defining additional derived data

Using derived data

Viewing derived data definitions

Saving derived data

Deleting derived data definitions