AtScale Semantic Modeling Language

Overview and Property Descriptions

ATSCALE

Introduction

The Semantic Modeling Language (SML) is an AtScale-provided modeling language that data engineers can use to create projects and models to be used within AtScale. This enables you to develop models programmatically and store them in an AtScale-connected Git repository. You can then deploy your models directly to AtScale and connect to them from your preferred business intelligence tool.

✏️**Note:** Currently, AtScale only supports integration with GitHub.

Semantic Modeling Language

Once you have your Git environment set up, you can start using the Semantic Modeling Language (SML) to programmatically create projects and models.

AtScale requires that each of your projects be stored in a single Git repository: one repository = one project. Each project repository must be structured as follows:

- `calculations/`
- `connections/`
- `datasets/`
- `dimensions/`
- `metrics/`
- `models/`
- `row_security/`
- `atscale.yml`

The following sections describe the different object types that SML supports, as well as the properties available for each:

- atscale.yml
- Models
- Metrics
- Calculations
- Connections
- Datasets
- Dimensions
- Row Security

atscale.yml

`atscale.yml` is the control file for an AtScale repository. It contains all repository-level definitions, such as the repository name and settings for building aggregates. Each project repository must contain an `atscale.yml` file at the root level.

✏️**Note:** Some properties can appear in both `atscale.yml` and model files. Those defined in model files override their counterparts in `atscale.yml`.

`atscale.yml` supports the following properties.

Property	Type	Required	Description
`unique_name`	string	Y	The name of the repository. This must be unique across all repositories and subrepositories.
`object_type`	const	Y	The type of object defined by the file. For `atscale.yml`, this must be `catalog`.
`label`	string	Y	The name of the repository, as it appears in AtScale. This value does not need to be unique.
`as_version`	number	Y	The version of SML being used.
`aggressive_agg_promotion`	boolean	Y	Enables/disables aggressive aggregate promotion for the repository. When enabled, all aggregates referenced by a query are considered for promotion, regardless of whether a join to other non-preferred or non-aggregate datasets was required. Supported values: ● `true` ● `false`
`build_speculative_aggs`	boolean	Y	Enables/disables speculative aggregates for the repository. When enabled, the AtScale engine automatically creates aggregate tables that it anticipates being useful based on your models. These are intended to improve the performance of queries from client BI tools faster than with demand-defined aggregates alone.

			✏️**Note:** In AtScale, speculative aggregates are also called prediction-defined aggregates. Supported values: • `true` • `false`
`dataset_properties`	object	N	Defines dataset properties to use within the repository. Supported properties: • `allow_aggregates`: Boolean, optional. Enables the AtScale engine to create aggregates for datasets in the project. • `allow_local_aggs`: Boolean, optional. Enables local aggregation for datasets in the project. • `allow_peer_aggs`: Boolean, optional. Enables aggregation on data derived from datasets in data warehouses that are different from the source dataset. • `allow_preferred_aggs`: Boolean, optional. Enables AtScale to promote aggregates from the model to the preferred aggregate storage location. Specify the `unique_name` of the dataset followed by the properties and values you want to set for it at the repository level. For example: Unset `dataset1:` `  create_hinted_aggregate: true` ✏️**Note:** Dataset properties are typically defined at the repository level, in the `atscale.yml` file; however, datasets used by a specific model (typically fact datasets) can have properties defined within the model itself. For more information, see Models.

Models

Model files define AtScale models. In AtScale, a model is a metadata layer that overlays a multi-dimensional model format on top of the datasets stored in a connected database. The model is virtual, meaning the data is not moved or processed up front. Instead, it contains the logic about how to process and optimize the data at query runtime.

✏️**Note:** Some properties can appear in both `atscale.yml` and model files. Those defined in model files override their counterparts in `atscale.yml`.

Model files support the following properties.

Property	Type	Required	Description
`unique_name`	string	Y	The unique name of the model. This must be unique across all repositories and subrepositories.
`object_type`	const	Y	The type of object defined by the file. For models, the value of this property should be `model`.
`label`	string	Y	The name of the model, as it appears in AtScale. This value does not need to be unique.
`relationships`	array	Y	Defines fact relationships specific to the model. For more information, see Model: relationships property. If you do not want to add relationships to the model, the value of this property must be `[]`. For example: `relationships: []` ✏️**Note:** These relationships are separate from those defined at the dimension level — relationships at the model level involve fact datasets, while those at the dimension level do not. For more information, see Appendix A: Relationships in AtScale Models.
`metrics`	array	Y	A list of references to metrics and calculations used in the model.

			Supported properties: ● `unique_name`: String, required. The unique name of the metric or calculation. This must be unique within the model file. ● `folder`: String, optional. The name of the folder in which the metric/calculation is displayed in BI tools. If your model has a lot of metrics/calculations, folders are a good way to organize them. ✏️**Note:** If you do not want to add any metrics to the model, the value of this property must be `[]`. For example: `metrics: []`
`description`	string	N	A description of the model.
`dimensions`	array	N	A list of references to degenerate dimensions defined on a specific fact dataset in the model.
`perspectives`	array	N	Defines perspectives for the model. For more information, see Model: perspectives property.
`drillthroughs`	array	N	Defines drillthroughs for the model. For more information, see Model: drillthroughs property.
`aggregates`	array	N	Defines user-defined aggregates for the model. For more information, see Model: aggregates property.
`partitions`	array	N	Defines partitions for the model. For more information, see Model: partitions property.
`dataset_properties`		N	Defines dataset properties that are specific to the model, rather than the repository. Supported properties: ● `allow_aggregates`: Boolean, optional. Enables the AtScale engine to create aggregates for datasets in the project. ● `allow_local_aggs`: Boolean, optional. Enables local aggregation for

			datasets in the project.
			● `allow_peer_aggs`: Boolean, optional. Enables aggregation on data derived from datasets in data warehouses that are different from the source dataset.
			● `allow_preferred_aggs`: Boolean, optional. Enables AtScale to promote aggregates from the model to the preferred aggregate storage location.
			
			Specify the name of the dataset followed by the properties and values you want to set for it at the model level. For example:
			
			Unset
			`dataset1:`
			`    create_hinted_aggregate: true`

Model: relationships property

The `relationships` property in a model file defines the relationships between the model's fact datasets and first order dimensions. These are called fact relationships.

✏️**Note:** Relationships defined at the model level are different from those defined at the dimension level, which do not include fact datasets. For more information, see Dimensions: relationships property and Appendix A: Relationships in AtScale Models.

✏️**Note:** Degenerate dimensions have relationships to the fact datasets on which they are based. However, these dimensions do not need a `relationships` property as they are created by referencing the fact dataset columns directly.

The `relationships` property of a model file supports the following properties.

Property	Type	Required	Description
`unique_name`	string	Y	The unique name of the relationship. This must be unique within the model file.
`from`	object	Y	Defines the side of the relationship that contains the physical fact dataset. Typically, this is a join column in the fact dataset.

			Supported properties: ● `dataset`: String, required. The physical fact dataset you want to link to a dimension. ● `join_columns`: Array, required. The columns within the `dataset` that you want to use as join columns.
`to`	object	Y	Defines the dimension that the `from` dataset is linked to. Supported properties: ● `dimension`: String, required if `row_security` is undefined. The name of the dimension to which the `from` dataset is joined. ● `level`: String, required if `row_security` is undefined. The `unique_name` of the level attribute within the `dimension` to use for the relationship. ● `row_security`: String, required if `dimension` and `level` are undefined. For security relationships, the row security object that the `from` dataset is joined to.
`role_play`	string	N	For role-playing relationships only. Defines the role-playing template for the relationship. The role-playing template is the prefix and/or suffix that is added to every attribute in the role-played dimension. This value must be in one of the following formats (including quotation marks): ● **Prefix:** `“<prefix> {0}”` ● **Suffix:** `“{0} <suffix>”` ● **Prefix and suffix:** `“<prefix> {0} <suffix>”` For example, if you wanted to use the prefix **Order**, you would set `role_play` to `“Order {0}”`.

Model: perspectives property

Perspectives are deployable subsets of the data model. They are meant to make it easier for analysts to query only the subset of data that is relevant to their purposes or responsibilities. Rather than provide analysts with the entire data model, you can make specific dimensions, hierarchies, levels, secondary attributes, measures, and calculated measures invisible to them.

The `perspectives` property in a model file supports the following properties.

Property	Type	Required	Description
`unique_name`	string	Y	The unique name of the perspective. This must be unique within the model file.
`metrics`	array	N	A list of the specific metrics and calculations available in the perspective.
`dimensions`	array	N	A list of the specific dimensions and their hierarchies available in the perspective. Supported properties: ● `name`: String, required. The name of the dimension to include in the perspective. ● `hierarchies`: Array, optional. A list of the specific hierarchies within the `name` dimension to include in the perspective. Supported properties: ○ `name`: String, required. The name of the hierarchy. ○ `levels`: Array, optional. A list of the levels within the hierarchy to include in the perspective. ● `secondaryattributes`: Array, optional. A list of the dimension's secondary attributes to include in the perspective.

Model: drillthroughs property

In BI tools, a drillthrough enables you to view detailed information about a specific cell within a visualization as needed. This provides an alternative to including lots of fine-grained attributes in large pivot tables, which can result in performance issues. Moving these attributes to drillthroughs means they are only returned if a user requests them for a specific cell, rather than for the entire table.

In an SML model, you can define drillthroughs that include the specific level of detail to return for these types of queries.

The `drillthroughs` property in a model file supports the following properties.

Property	Type	Required	Description
`unique_name`	string	Y	The unique name of the drillthrough. This must be unique within the model file.
`metrics`	array	Y	A list of the metrics to include in the drillthrough.
`notes`	string	N	Notes about the drillthrough.
`attributes`	array	N	A list of the specific attributes to include in the drillthrough. Supported properties: ● `name`: String, required. The name of the attribute to include in the drillthrough. ● `dimension`: String, required. The dimension that the attribute defined by `name` appears in.

Model: aggregates property

The `aggregates` property in a model file enables you to add user-defined aggregates (UDAs).

In general, AtScale recommends relying on the aggregate tables automatically generated by the AtScale engine. However, there are cases that are not covered by system-defined aggregates. For example:

- **Metrics on dimensions:** The AtScale engine does not generate aggregate tables for metrics that are local to a dimension only (a secondary metrical attribute in the model).
- **Non-additive metrics:** The AtScale engine does not generate aggregate tables for non-additive metrics, which are useful for distinct counts. This is because such an aggregate table defined for one query would not be usable by other queries.

If you require aggregate tables that contain these types of dimensional attributes or metrics, you should define your own manually using the `aggregates` property.

The `aggregates` property in a model file supports the following properties.

Property	**Type**	**Required**	**Description**
`unique_name`	string	Y	The unique name of the aggregate. This must be unique within the model file.
`label`	string	Y	The name of the aggregate, as it appears in AtScale. This value does not need to be unique.
`target_connection`	string	Y	The database that the AtScale engine writes the aggregate table to.
`metrics`	array	Y	A list of the metrics and calculations to include in the aggregate definition.
`attributes`	array	N	A list of the dimension attributes to include in the aggregate definition. Supported properties: ● `name`: String, required. The name of the dimension attribute to include. These values are used to group the summarized metric data in the resulting aggregate table. ● `dimension`: String, required. The dimension to which the attribute defined by `name` belongs. ● `partition`: String, optional. Adds a partition to the aggregate, and determines whether it should be defined on the key column, name column, or both. Supported values: `name, key, name+key` ● `distribution`: String, optional. The distribution keys to use when creating the aggregate table.

Model: partitions property

The `partitions` property in a model file enables you to create prioritized partitioning hints that the AtScale engine uses to create partitioned aggregate tables. The actual partitioning scheme used by the engine depends on a number of factors, including:

- Whether the aggregate includes a column that matches a partition hint.
- Whether AtScale statistics suggest that partitioning would be worthwhile.

- Whether the target data warehouse supports table partitioning.

Within SML, all partitions used in a model are defined in the model file itself.

The `partitions` property in a model file supports the following properties.

Property	Type	Required	Description
`unique_name`	string	Y	The unique name of the partition. This must be unique within the model file.
`dimension`	string	Y	The dimension that contains the `attribute` the partition is based on.
`attribute`	string	Y	The attribute that the partition is based on.
`type`	string	Y	Determines whether the partition is defined on the name column, key column, or both. Supported values: • `name` • `key` • `name+key`

Metrics

Metric files define measures to be used in your repository. A measure is a numeric value representing a summarized (or aggregated) dataset metric, such as the sum of sales or average order quantity. Metrics always result from an aggregate calculation applied to one or more columns of a fact dataset.

✏️**Note:** SML uses the term *metric* to refer to *measures* in AtScale.

AtScale supports the following types of metrics:

- **Additive:** Metrics whose values can be summarized for any dimension attribute of the model and then combined consistently.
- **Non-additive:** Metrics whose values cannot be summed across any dimensional groupings using basic addition, since this would typically produce an inaccurate result. The most common example of a non-additive metric is a distinct count of an attribute value.
- **Semi-additive:** Metrics whose values can be summarized for some dimensions in a model, but not all. Ratios such as average are also considered semi-additive metrics.

✏️**Note:** AtScale also supports calculated metrics, which in SML are defined in calculation files. For more information, see Calculations.

Metric files support the following properties.

Property	**Type**	**Required**	**Description**
`unique_name`	string	Y	The unique name of the metric. This must be unique across all repositories and subrepositories.
`object_type`	string	Y	The type of object defined by the file. For metrics, the value of this property should be `metric`.
`label`	string	Y	The name of the metric, as it appears in AtScale. This value does not need to be unique.
`calculation_method`	string	Y	The method used to aggregate query results for the metric. Supported values: • `average` • `count distinct` • `count non-null` • `estimated count distinct` • `maximum` • `minimum` • `percentile` • `stddev_pop` • `stddev_samp` • `sum` • `var_pop` • `var_samp` The calculation method you can use depends on the type of metric you're creating: • Semi-additive: `average`, `sum`, `minimum`, or `maximum` • Non-additive: `count distinct` or `percentile` • Additive: All other options
`dataset`	string	Y	The source dataset that contains the column the metric is based on.
`column`	column	Y	The specific column within the `dataset` that

			the metric is based on.
`description`	string	N	A description of the metric.
`semi_additive`	object	N	Defines the metric as a semi-additive metric. Supported properties: ● `position`: String, required. Determines whether the metric is First Non-Empty or Last Non-Empty. Supported values: `first`, `last` ● `dimension`: String, required. The dimension with which the semi-additive metric is associated. ● `hierarchy`: String, required. The hierarchy with which the semi-additive metric is associated. ● `level`: String, required. The level with which the semi-additive metric is associated.
`compression`	number	N	Only for non-additive metrics using a `calulation_method` of `percentile`. Defines the compression score the AtScale engine uses when estimating percentile values for query results. You can specify a value between 1 – 50,000. Using a higher compression score yields more accurate query results but requires more memory from the engine to process. You may need to run tests to determine the right level of compression for your needs. ✏️**Note:** In AtScale, `compression` is referred to as **Quality**.
`named_quantiles`	string	Required if `calculation_method` is `percentile`	Only for non-additive metrics using a `calulation_method` of `percentile`. Defines the quantile to use for query results. Supported values: ● `quartiles` ● `median` ● `deciles`
`format`	string	N	The format in which query results are

			returned. You can use one of AtScale's built-in named formats or a custom format string. Supported named formats: ● `fixed` ● `general number` ● `none` ● `percent` ● `scientific` ● `standard` Custom format strings should be in quotes and contain one to four sections, separated by semicolons. For example: `"$#,##0.00"`
`unrelated_dimensions_handling`	string	N	Determines how the AtScale engine behaves when all of the following conditions are true: ● A client queries a model that contains multiple fact datasets. ● The data in each fact dataset are at a different level of granularity than the data in the other fact datasets. ● The query references dimensions that are not related to the metrics being queried. Supported values: ● `error:` AtScale rejects the query and returns an error message. ● `empty:` AtScale displays empty cells in the query results. ● `repeat:` In the query results, AtScale repeats the values for the metric at a level of aggregation that is determined from the shared dimensions in the query.
`is_hidden`	boolean	N	Determines whether the metric appears in BI tools. Supported values: ● `true` ● `false`

Calculations

Calculation files define custom MDX expressions for creating calculated metrics in AtScale. They can be used to combine, evaluate, or manipulate other metrics defined in the model. For example, you can do simple math operations to combine metrics, or simple comparison operations to return a given metric value when certain conditions are met.

In SML, calculation files are a subset of metrics. The separation of calculation metrics from other types enables you to easily create boilerplate calculations that can be used across multiple metrics.

✏️**Note:** In AtScale, calculations are referred to as *calculated measures*.

Calculation files support the following properties.

Property	**Type**	**Required**	**Description**
`unique_name`	string	Y	The unique name of the calculation. This must be unique across all repositories and subrepositories.
`object_type`	string	Y	The type of object defined by the file. For calculations, the value of this property should be `metric_calc`.
`label`	string	Y	The name of the calculation as it appears in AtScale. This value does not need to be unique.
`expression`	string	Y	The MDX expression to use for the calculation. This expression must be written in MDX syntax, surrounded by quotes (“). Additionally, it can only operate on existing metrics in the model, and must return a numeric value. ✏️**Note:** AtScale only supports a small subset of MDX functions and operators.
`description`	string	N	A description of the calculation.
`format`	string	N	The format of the values returned by the calculation. You can use one of AtScale’s built-in named formats or a custom string

			format. Supported named formats: ● `fixed` ● `general number` ● `none` ● `percent` ● `scientific` ● `standard` Custom format strings should be in quotes (“) and contain one to four sections, separated by semicolons. For example: `"$#,##0.00"`
`is_hidden`	boolean	N	Determines whether the calculation is visible in BI tools. Supported values: ● `true` ● `false`

Connections

Connection files define database connections and schemas for the repository. These are required to import fact and dimension datasets into your repository.

Each connection file should define a single database connection *and* its schema. If you need to use additional schemas for the same database, each must be defined in a separate connection file.

Connection files support the following properties.

Property	**Type**	**Required**	**Description**
`unique_name`	string	Y	A unique name for the database and the schema. This must be unique across all repositories and subrepositories.
`object_type`	const	Y	The type of object defined by this file. For connections, this value must be `connection`.
`label`	string	Y	The name of the database connection as it appears in AtScale. This value does not need to be unique.

`as_connection`	string	Y	The name of the database connection itself, excluding the schema.
`database`	string	Y	The source database used for this connection.
`schema`	string	Y	The source schema used for this connection.

Datasets

Dataset files define datasets to use in the repository. Each dataset file in your repository must correspond to either a physical table/view in your database, or the results of a SELECT statement.

✏️**Note:** Dataset files must define *all* columns in the physical tables they reference, and can therefore be quite large. Because of this, AtScale recommends sharing these files across repositories.

Dataset files support the following properties.

Property	**Type**	**Required**	**Description**
`unique_name`	string	Y	The unique name of the dataset. This must be unique across all repositories and subrepositories.
`object_type`	const	Y	The type of object defined by the file. For datasets, the value of this property must be `dataset`.
`label`	string	Y	The name of the dataset, as it appears in AtScale. This value does not need to be unique.
`connection_id`	string	Y	The `unique_name` of the connection object that defines the database and schema in which the dataset is stored.
`sql`	string	Required if `table` is not provided	A SQL query used to pull data from a specific connection defined within the repository, similar to a database view. This determines whether the dataset file defines a query dataset.
`table`	string	Required	The name of the table in the `database` that

		if `sql` is not provided	the dataset is based on.
`columns`	array	Y	Defines the columns available in the dataset. For more information, see Dataset: columns property.
`description`	string	N	A description of the dataset.
`immutable`	boolean	N	Determines whether the dataset changes often or not. The AtScale engine uses this information when running incremental builds of aggregates that use joins on dimensions that do not change often.

Dataset: columns property

The columns property within a dataset file defines the columns available in the dataset.

✏️**Note:** You should define *all* columns available in the dataset. This is especially important for dataset files that are shared across multiple repositories.

The `columns` property within a dataset file supports the following properties.

Property	**Type**	**Required**	**Description**
`name`	string	Y	The name of the column.
`data_type`	string	Required unless column is a `map`	The data type of the values within the column. Supported values: ● `string` ● `int` ● `long` ● `bigint` ● `tinyint` ● `float` ● `double` ● `decimal` ● `decimal(x,y)` ● `number` ● `number(x,y)` ● `numeric(x,y)` ● `boolean`

			<ul><li>`date`</li><li>`datetime`</li><li>`timestamp`</li></ul>
`sql`	string	N	Defines the column as a calculated column. Calculated columns enable you to add simple data transformations to the dataset. These can be used as the basis of model attributes, just like any other dataset column. The value of this property should be a valid SQL statement that can be run as part of the `SELECT` list of a query. The SQL statement is passed directly to the underlying database when the query runs, so it must be in a syntax that is supported by your chosen engine. If you want to run the query on other types of databases, use the `dialects` property to define additional dialects for it to run in.
`map`	object	N	Defines a map used to create a calculated column. Supported properties:<ul><li>`field_terminator`: String, required. The delimiter used to separate the key:value pairs. This must be in quotes (").</li><li>`key_terminator`: String, required. The delimiter used to separate the individual keys from their values. This must be in quotes (").</li><li>`key_type`: String, required. The data type of the map's keys.</li><li>`value_type`: String, required. The data type of the map's values.</li></ul>The mapped columns are defined as separate columns within the `dataset` file. Each of these must have the `parent_column` property.
`parent_column`	string	Required for mapped	For mapped columns only. Specifies the `map` column used to create this column.

		columns	

Dimensions

Dimension files define the dimensions used in the model. A *dimension* is a logical collection of attributes that are bound to specific columns in a source dataset. These attributes are in turn used to group and filter metric data at query time.

AtScale supports the following types of dimensions:

- **Normal:** Dimensions that are based on a dataset. All data for a normal dimension is normalized into a single table or view. There are two types of normal dimensions:
    - **Standard:** Can have any type of hierarchy.
    - **Time:** Must have a time hierarchy
- **Degenerate:** A dimension that is based on one or more columns in a fact dataset.
- **Shared degenerate:** A dimension that is based on one or more columns that are common to two or more fact datasets.
- **Snowflake:** A logical dimension that is composed of multiple underlying physical datasets.
- **Many-to-many:** Also called multi-valued. This is when a fact dataset row refers to more than one row in a dimension dataset. In AtScale, this is modeled by defining a dimensional bridge or junction table to resolve the many-to-many relationship.

Dimension files support the following properties.

Property	Type	Required	Description
`unique_name`	string	Y	The unique name of the dimension. This must be unique across all repositories and subrepositories.
`object_type`	const	Y	The type of object defined by the file. For dimensions, this value should be `dimension`.
`label`	string	Y	The name of the dimension, as it appears in AtScale. This value does not need to be unique.
`hierarchies`	array	Y	Defines the dimension’s hierarchies. For more information, see Hierarchies.
`level_attributes`	array	Y	Defines the level attributes in the dimension. For more information, see Dimensions: level_attributes property.

`relationships`	array	N	Defines the relationships in the dimension. For more information, see [Dimensions: relationships property]().
`calculation_groups`	array	N	Defines the calculation groups in the dimension. For more information, see [Dimensions: calculation_groups property]().
`description`	string	N	A description of the dimension.
`type`	enum	N	The type of dimension defined by this file. Supported values: ● `standard`: Can have any type of hierarchy. ● `time`: Must have a time hierarchy.

Dimensions: hierarchies property

The `hierarchies` property in a dimension file defines the hierarchies within the dimension.

Hierarchies organize the dimension attributes into categories or levels, where each level is a subdivision of the level above. Every logical dimension you create has at least one hierarchy with at least one level.

The `hierarchies` property within a dimension file supports the following properties.

Property	**Type**	**Required**	**Description**
`unique_name`	string	Y	The unique name of the hierarchy. This must be unique within the dimension.
`label`	string	Y	The name of the hierarchy, as it appears in AtScale. This value does not need to be unique.
`levels`	array	Y	Defines the levels within the hierarchy. You can include as many levels as needed in the list. Supported properties: ● `unique_name`: String, required. Specifies the unique name of the level. This must be unique within the dimension.

			• `time_unit`: String, for time dimensions only. The unit of time to use. Supported values: `year, halfyear, trimester, quarter, month, week, day, hour, minute, second, undefined` • `secondary_attributes`: Array, optional. Defines the secondary attributes for the level. For the full list of properties this can include, see Diemnsions: hierarchies: levels: secondary_attributes property. • `aliases`: Array, optional. Defines secondary attributes that can be used as aliases for specific hierarchy levels within BI tools. For more information, see Dimensions: hierarchies: levels: aliases property. • `metrics`: Array, optional. Defines metrics for the level. For more information, see Dimensions: hierarchies: levels: metrics property.
`description`	string	N	A description of the hierarchy.
`folder`	string	N	The name of the folder in which to display this hierarchy in BI tools. If your model has a lot of dimensional hierarchies, folders are a good way to organize them.
`filter_empty`	string	N	Configures the join behavior for the hierarchy, which determines how empty values are handled in client BI tools. The value you specify must be in quotes (“). Supported values: • `yes`: Query results in BI tools only include members that join to the fact dataset (inner join behavior). Members with no matching entries in the fact dataset are still included if the client BI tool requests them. • `no`: Query results include all members of the dimension, even those that have no matching entries in the fact dataset (outer join behavior). This occurs unless the client BI tool specifically

			requests to have these values filtered out. ● `always`: Query results only include members that join to the fact dataset (inner join behavior). This typically provides the best performance.
`default_member`	string	N	Defines a member of the hierarchy to use as the default filter for MDX queries on the hierarchy. The value must be formatted as an MDX expression and must be in quotes (“). ✏️**Note:** You cannot specify secondary attributes as default dimension members.

Dimensions: hierarchies: levels: secondary_attributes property

Secondary attributes are dimensional attributes that are not the dimension’s key, and are not part of a hierarchy.

AtScale supports the following types of secondary attributes:

- **Dimensional:** Provides an independent "dimensional" attribute for grouping metric data. This is the default type of secondary attribute.
- **Level alias:** Enables the creation of tabular reports that select hierarchical expressions without forcing the user to drill down a hierarchy.

✏️**Note:** Secondary attributes cannot be used to create relationships between datasets and dimensions.

Within SML, secondary attributes are defined by the `hierarchies` > `levels` > `secondary_attributes` property within a dimension file. You can define as many secondary attributes as needed in the list.

The `secondary_attributes` property within a dimension hierarchy level supports the following properties.

Property	Type	Required	Description
`unique_name`	string	Y	The unique name of the secondary attribute. This must be unique within the dimension.
`label`	string	Y	The name of the secondary attribute, as it appears in AtScale. This value does not need

			to be unique.
`dataset`	string	Y	The dataset that contains the `key_columns` the secondary attribute is based on.
`name_column`	string	Y	The dataset column that the attribute is based on.
`key_columns`	array	Y	A list of the key columns that a dimension attribute is based on. If the attribute has a compound key, you should specify all columns that make up the key as a list.
`sort_column`	string	N	The column used to sort the attribute's values in result sets. ✏️**Note:** This only applies to MDX queries.
`allowed_calcs_for_dma`	array	N	A list of the calculation types that can be used to create dimensionally modified aggregates for the secondary attribute. ✏️**Note:** When working with a time dimension, you can only define calculation types if the `time_unit` property for the level is set to `day` or longer.
`exclude_from_dim_agg`	boolean	N	Excludes this attribute from system generated dimension-only aggregates. This is useful if the attribute contains a large number (millions) of distinct values that you don't want to aggregate.
`exclude_from_fact_agg`	boolean	N	Excludes this attribute from system generated fact-based aggregates. This is useful if the attribute contains a large number (millions) of distinct values that you don't want to aggregate.
`custom_empty_member`	array	N	Defines a custom empty member for the attribute. This feature allows fact data with missing or invalid foreign key values to be isolated and independently aggregated from those with valid foreign key values. Because fact records with invalid foreign keys are aggregated separately from records referencing valid

			dimension members, analysts can easily spot data integrity problems and further investigate them. Use this feature to ensure that un-joinable values are included in query results and aggregated under a specially designated dimension member called the Custom Empty Member. Supported properties: ● `key`: Array, required. A list of the empty member values to use for key fields. ● `name`: String, required. The empty member value to use for name fields. ● `sort`: String, optional. The empty member value to use for the attribute's sort column, if one is specified.
`description`	string	N	A description of the secondary attribute.
`is_hidden`	boolean	N	Determines whether the attribute is visible in BI tools. Supported values: ● `false` (default) ● `true`
`folder`	string	N	The name of the folder in which the attribute is displayed in BI tools.
`contains_unique_names`	boolean	N	Determines whether each member of this attribute has a unique name. Do not enable this functionality if two members have different keys but the same name. Supported values: ● `true` ● `false`

Dimensions: hierarchies: levels: aliases property

The `aliases` property defines secondary attributes to use as aliases for specific levels within a hierarchy. These are useful in BI tools, as they enable the user to select a specific level without having to navigate through the hierarchy it belongs to. You can include as many aliases as needed in the list.

The `aliases` property within a dimension hierarchy level supports the following properties.

Property	Type	Required	Description
`unique_name`	string	Y	The unique name of the alias. This must be unique within the dimension.
`label`	string	Y	The name of the alias, as it appears in AtScale and BI tools. This value does not need to be unique.
`dataset`	string	Y	The source dataset that contains the column that the alias is based on.
`name_column`	string	Y	The dataset column that the alias is based on.
`sort_column`	string	N	The column used to sort the values in result sets. This applies to MDX queries only (queries received through the XMLA interface).
`description`	string	N	A description of the alias.
`is_hidden`	boolean	N	Determines whether the alias is visible in BI tools. Supported values: ● `true` ● `false`
`exclude_from_dim_agg`	boolean	N	Excludes this alias from system generated dimension-only aggregates. This is useful if the alias contains a large number (millions) of distinct values that you don't want to aggregate. Supported values: ● `true` ● `false`
`exclude_from_fact_agg`	boolean	N	Excludes this alias from system generated fact-based aggregates. This is useful if the alias contains a large number (millions) of distinct values that you don't want to aggregate.

			Supported values: ● `true` ● `false`
`cusom_empty_member`	object	N	Defines custom empty member values for the alias. This feature allows fact data with missing or invalid foreign key values to be isolated and independently aggregated from those with valid foreign key values. Because fact records with invalid foreign keys are aggregated separately from records referencing valid dimension members, analysts can easily spot data integrity problems and further investigate them. Use this feature to ensure that un-joinable values are included in query results and aggregated under a specially designated dimension member called the Custom Empty Member. Supported properties: ● `key`: Array, required. A list of the empty member values to use for key fields. ● `name`: String, required. The empty member value to use for name fields. ● `sort_name`: String, optional. The empty member value to use for the alias's `sort_column`, if one is specified.
`format`	string	N	The format in which query results are returned. You can use one of AtScale's built-in named formats or a custom format string. Supported named formats: ● `fixed` ● `general number` ● `none` ● `percent` ● `scientific` ● `standard`

			Custom format strings should be in quotes and contain one to four sections, separated by semicolons. For example: `"$#,##0.00"`
`folder`	string	N	The name of the folder in which the alias appears in BI tools.

Dimensions: hierarchies: levels: metrics property

The metrics property of a dimension level defines secondary metrical attributes for the dimension, which behave like metrics in a very limited context of the data model.

✏️**Note:** This feature is experimental and must be enabled within AtScale by an admin.

The `metrics` property within a dimension hierarchy level supports the following properties.

Property	Type	Required	Description
`label`	string	Y	The name of the secondary metrical attribute, as it appears in AtScale. This value does not need to be unique.
`unique_name`	string	Y	The unique name of the secondary metrical attribute. This must be unique within the dimension.
`dataset`	string	Y	The source dataset that contains the column that the secondary metrical attribute is based on. This should be the dimension dataset name.
`column`	string	Y	The column within the `dataset` that the secondary metrical attribute is based on.
`calculation_method`	string	Y	The calculation to apply to the data. Supported values: ● `average` ● `count distinct` ● `count non-null` ● `estimated count distinct` ● `maximum` ● `minimum` ● `percentile` ● `stddev_pop`

			<ul><li>`stddev_samp`</li><li>`sum`</li><li>`var_pop`</li><li>`var_samp`</li></ul>
`description`	string	N	A description of the secondary metrical attribute.
`is_hidden`	boolean	N	Determines whether the secondary metrical attribute is visible in BI tools. Supported values:<ul><li>`true`</li><li>`false`</li></ul>
`folder`	string	N	The name of the folder in which the secondary metrical attribute appears in BI tools.
`format`	string	N	The format in which query results are returned. You can use one of AtScale's built-in named formats or a custom format string. Supported named formats:<ul><li>`fixed`</li><li>`general number`</li><li>`none`</li><li>`percent`</li><li>`scientific`</li><li>`standard`</li></ul>Custom format strings should be in quotes and contain one to four sections, separated by semicolons. For example: `"$#,##0.00"`
`exclude_from_dim_agg`	boolean	N	Excludes this secondary metrical attribute from system generated dimension-only aggregates. This is useful if the secondary metrical attribute contains a large number (millions) of distinct values that you don't want to aggregate. Supported values:<ul><li>`true`</li><li>`false`</li></ul>
`exclude_from_fact_agg`	boolean	N	Excludes this secondary metrical attribute from system generated fact-based

			aggregates. This is useful if the secondary metrical attribute contains a large number (millions) of distinct values that you don't want to aggregate. Supported values:<ul><li>`true`</li><li>`false`</li></ul>
`custom_empty_member`	object	N	Defines custom empty member values for the secondary metrical attribute. This feature allows fact data with missing or invalid foreign key values to be isolated and independently aggregated from those with valid foreign key values. Because fact records with invalid foreign keys are aggregated separately from records referencing valid dimension members, analysts can easily spot data integrity problems and further investigate them. Use this feature to ensure that un-joinable values are included in query results and aggregated under a specially designated dimension member called the Custom Empty Member. Supported properties:<ul><li>`key`: Array, required. A list of the empty member values to use for key fields.</li><li>`name`: String, required. The empty member value to use for name fields.</li><li>`sort`: String, optional. The empty member value to use for the secondary metrical attribute's sort column, if one is specified.</li></ul>
`unrelated_dimensions_handling`	enum	N	Determines how the AtScale engine behaves when all of the following conditions are true:<ul><li>A client queries a model that contains multiple fact datasets.</li><li>The data in each fact dataset are at a different level of granularity than the data in the other fact datasets.</li><li>The query references dimensions that are not related to the metrics being</li></ul>

			queried. Supported values: ● `error`: AtScale rejects the query and returns an error message. ● `empty`: AtScale displays empty cells in the query results. ● `repeat`: In the query results, AtScale repeats the values for the secondary metrical attribute at a level of aggregation that is determined from the shared dimensions in the query.

Dimensions: level_attributes property

Level attributes are attributes associated with a particular dimension hierarchy. Every hierarchy has a key level attribute, which is the most granular representation of the dimension's data. Only level attributes can be used to define relationships between datasets and other dimensions.

Within SML, level attributes are defined by the `level_attributes` property of a dimension file.

The `level_attributes` property of a dimension file supports the following properties.

Property	Type	Required	Description
`unique_name`	string	Y	The unique name of the level attribute. This must be unique within the dimension.
`label`	string	Y	The name of the level attribute, as it appears in AtScale. This value does not need to be unique.
`dataset`	string	Y	The source dataset that contains the columns that this level attribute is based on.
`name_column`	string	Y	The column whose values appear for this level attribute in BI tools. For example, the key may be a product ID number, but you want users to see product names instead.
`key_columns`	array	Y	The dataset column that the level attribute is based on. If the level attribute has a compound key, list all columns that make up

			the key. If the key consists of one column, the values in that column must be unique. If the key is a compound key, the columns together must provide unique values.
`description`	string	N	A description of the level attribute.
`is_hidden`	boolean	N	Determines whether the level attribute is visible in BI tools. Supported values: ● `true` ● `false`
`is_unique_key`	boolean	N	Determines whether the [`key_columns`]() values are unique for each row. Supported values: ● `true`: The key column values are unique for each row. The join behavior considers the first matching row at query runtime. ● `false`: The key column values are multi-valued. The join behavior considers all matching rows at query runtime. ✏️**Note:** Setting this value to `true` is equivalent to declaring the key to be a primary key. The AtScale engine uses this property as input when joining rows from this level attribute to other datasets in the model.
`contains_unique_names`	boolean	N	Determines whether each member of this level attribute has a unique name. Do not enable this functionality if two members have different keys but the same name. Supported values: ● `true` ● `false`
`exclude_from_dim_agg`	boolean	N	Excludes this level attribute from system generated dimension-only aggregates. This is useful if the level attribute contains a large number (for example, in the millions) of

			distinct values that you don't want to aggregate. Supported values: ● `true` ● `false`
`exclude_from_fact_agg`	boolean	N	Excludes this level attribute from system generated fact-based aggregates. This is useful if the level attribute contains a large number (for example, in the millions) of distinct values that you don't want to aggregate. Supported values: ● `true` ● `false`
`sort_column`	string	N	Defines the column to sort query results on. By default, this is the `name_column`; however, you can optionally use this property to specify a different column. ✏️**Note:** This only applies to MDX queries (queries received through the XMLA interface).
`allowed_calcs_for_dma`	array	N	A list of the calculations that can be used when creating dimensionally modified aggregates for the level attribute.
`folder`	string	N	The name of the folder in which this level attribute appears in BI tools.

Dimensions: relationships property

The `relationships` property in a dimension file defines the relationships to embedded and snowflake dimensions.

✏️**Note:** The relationships between the model's fact datasets and first order dimensions (fact relationships) are defined in model files.

For more information on relationships in AtScale, see Appendix A: Relationships in AtScale Models.

The `relationships` property in a dimension file supports the following properties.

Property	**Type**	**Required**	**Description**
`from`	object	Y	Defines the side of the relationship that contains the physical dataset that you want to connect to another dimension. Supported properties: ● `dataset`: String, required. The physical dataset you want to link to a dimension. ● `join_columns`: Array, required. The column(s) within the `dataset` that you want to use for the join. ● `hierarchy`: String, optional. The hierarchy within the dimension from which the relationship should originate. ● `level`: String, optional. The level within the `hierarchy` from which the relationship should originate. For `snowflake` relationships (as defined by the `type` property), you only need to define `dataset` and `join_columns`.
`to`	object	Y	Defines the dimension that the `from` dataset is linked to. Supported properties: ● `dimension`: String. The name of the dimension the `from` dataset is linked to. ● `level`: String, required if `row_security` is undefined. The key level within the dimension to use for the relationship. ● `row_security`: String, required if `level` is undefined. For security relationships, the row security object that the `from` dataset is linked to. For `snowflake` relationships (as defined by the `type` property), you only need to define `level`.
`type`	string	Y	Defines the relationship as either embedded

			or snowflake. Supported values: ● `embedded`: A secondary relationship, or one that connects a primary dimension to a secondary dimension. ● `snowflake`: A relationship that connects one of several underlying physical datasets together to create a snowflake dimension.
`role_play`	string	N	For role-playing relationships only. Defines the role-playing template for the relationship. The role-playing template is the prefix or suffix that is added to every attribute in the role-played dimension. You can also specify both a prefix and a suffix. This value must be in one of the following formats (including quotation marks): ● **Prefix:** `“<prefix> {0}”` ● **Suffix:** `“{0} <suffix>”` ● **Prefix and suffix:** `<“prefix> {0} <suffix>”` For example, if you wanted to use the prefix **Order**, you would set `role_play` to `“Order {0}”`.
`unique_name`	string	N	The unique name of the relationship. This must be unique within the dimension.

Dimension: calculation_groups property

The `calculation_groups` property in a dimension file defines calculation groups to use in the dimension.

Dimension calculation groups offer a simplifying alternative to calculated metrics by enabling the expression of boiler-plate calculations across multiple metrics. This feature defines calculations as dimension members and removes static references to individual measures.

The `calculation_groups` property in a dimension file supports the following properties.

Property	Type	Required	Description
`unique_name`	string	Y	The name of the calculation group. This must be unique within the dimension.
`description`	string	Y	A description of the calculation group.
`calculated_members`	array	Y	Defines the individual calculations in the group. Supported properties: ● `unique_name`: String, required. The name of the calculation. This must be unique within the dimension. ● `description`: String, required. A description of the calculation. ● `expression`: String, required. The MDX expression for the calculation. This must be in quotes. ● `format`: String, optional. The format for the calculation results. You can use one of AtScale's built-in named formats or a custom format string: ○ Supported named formats: `fixed`, `general number`, `none`, `percent`, `scientific`, `standard` ○ Custom format strings should be in quotes and contain one to four sections, separated by semicolons. For example: `"$#,##0.00"`
`folder`	string	N	The name of the folder in which the calculation group appears in BI tools.

Row Security

Row security files enable you to define security objects, which restrict access to data in a model. These restrictions can be configured at either the user or the group level. When users run queries against a model, AtScale uses the `row_security` as a runtime constraint.

Row security requires a separate dataset that maps user or group IDs to specific rows in a dimension or fact dataset. Each user or group can only access the data in rows that match the filter; for example, you can restrict a user's access to rows relating to specific countries only.

Once you create a security row object, you can use it to secure other dimensions and datasets in a model by creating a relationship from the dataset/dimension you want secured to the security row file. For more information, see Model: relationships property and Dimension: relationships property.

Row security files support the following properties.

Property	**Type**	**Required**	**Description**
`unique_name`	string	Y	The unique name of the security object. This must be unique across the repositories and all subrepositories.
`object_type`	const	Y	The type of object defined by the file. For row security files, the value of this property must be `row_security`.
`label`	string	Y	The name of the security object, as it appears in AtScale. This value does not need to be unique.
`dataset`	string	Y	The dataset that contains the user-to-attribute mappings determining which rows each user/group can access.
`filter_key_column`	string	Y	The column in the security dataset that defines the rows each user/group has access to.
`ids_column`	string	Y	The column of the security dataset that contains AtScale user/group IDs.
`id_type`	string	Y	Determines whether the IDs are for users or groups. Supported values: ● `user` ● `group`
`scope`	string	Y	Determines which queries the security constraint is applied to.

			Supported values: ● `related`: The security constraint is applied when the query selects any dimension or secondary attribute that has a path to the security dataset, as long as no fact table is used. The security constraint is *not* applied to dimension-only queries that select multiple dimensions related through a fact table. ● `fact`: The security constraint is applied to the same queries as the `related` option, as well as queries that include a measure from a fact table connected to the secure dimension. The security constraint is *not* applied to single-dimension-only queries that are related to the secured dimension via the fact table. However, multi-dimension-only queries do have security applied because they are joined using a synthetic measure from the fact table that relates them. ● `all`: The security constraint is applied to all queries, unless there is no path to the security dimension. This is the case with two separate fact tables, each with their own unrelated dimensions.
`description`	string	N	A description of the security object.
`use_filter_key`	boolean	N	Determines how AtScale enforces security. Supported values: ● `true`: The system first looks up the `filter_key_column` values using the user or goup's ID, and then uses those values as a constraint in a second query against the fact dataset or dimension. Some data warehouses perform better with this option. ● `false`: The system enforces security by joining with the security table.
`secure_totals`	boolean	N	Enables/disables the secure totals functionality.

			When enabled, the security restriction applies to the following: ● Subtotal measures of the secured hierarchy level or reachable attributes of higher levels. ● Queries that select secured fact tables (a `scope` of `all` or `fact`), but do not select the secured dimension. ● The grouping of the secured level. ● The secured level's secondary attributes. ● Attributes and nested dimensions that are reachable from hierarchy levels lower than the secured level. When secured totals is disabled, the security restriction only applies to the following: ● The grouping of the secured level. ● The secured level's secondary attributes. ● Attributes and nested dimensions that are reachable from hierarchy levels lower than the secured level. Supported values: ● `true` (default) ● `false`

Appendices

Appendix A: Relationships in AtScale Models

Relationships play an integral role in AtScale models — they define the links between physical datasets and logical dimensions. They are not modeled between two datasets directly. When you create a relationship in a model, you provide information that the AtScale engine can use to join the underlying tables at query time.

✏️**Note:** A dimension is not considered part of a model until it has a relationship to a fact dataset within the model (either directly or indirectly).

AtScale supports the following types of relationships:

- **One-to-many:** When modeling data in a star schema format, dimension-to-fact relationships are typically one-to-many. This means that each record in the fact dataset can link to one (and only one) record in the dimension dataset, but a record in the dimension dataset can be associated with many fact records.

- **Many-to-many:** Real-world use cases do not always align with the one-to-many star schema model. Some relationships can only be represented as a many-to-many relationship. This occurs when a fact dataset row can refer to more than one row in a dimension dataset. In AtScale, this is modeled by defining a dimensional bridge to resolve the many-to-many relationship.
- **Role-playing:** Whenever you create a relationship to a dimension, whether from a fact table to a dimension or from one dimension to another dimension, an instance of that dimension is added to the model. In some cases, the same dimension may be referenced in more than one context in the same model. A role-playing relationship is what differentiates multiple instances of the same dimension in a model.
- **Multi-fact:** A multi-fact model is when you want to analyze measures that originate from two different fact datasets. This is possible in AtScale, provided that both fact datasets have relationships to common dimensions.

Within SML, relationships can be defined at both the model and dimension levels:

- Relationships defined at the model level are strictly between fact datasets and dimensions.
- Relationships at the dimension level are between other types of datasets used in the model and dimensions.

For information on defining relationships in SML, see Model: relationships property and Dimensions: relationships property.