AltaSQL For Snowflake

AltaSQL Enterprise User Guide

Contents

Introduction

AltaSQL Enterprise is an upgrade in functionality to SELECT Discover.

If you have not used SELECT Discover, we recommend that you start with the AltaSQL SELECT Discover Quick Start and review the SELECT Discover Key Features video before progressing to the new functionality.

Enterprise provides four new categories of functionality:

- Automated generation of CREATE VIEW statements with column comments, Dynamic Data Masking Policies and column tags in the pre-SELECT column list
- Multiple Source Objects
- "Plug-n-Play" for Modular Construction of SELECT, CREATE VIEW and CTAS statement
- Generation of Common Table Expressions (CTEs)
- Generation of Create Table AS (CTAS) statements from View Definitions, including creating temporary tables
- Additional functions for creating expressions

Generating CREATE VIEW DDL

SBLD_VIEW FUNCTIONS

There are three functions for generating CREATE VIEWs statements corresponding to the Snowflake CREATE VIEW statements:

- `SBLD_VIEW` – generated DDL returns an error when executed if the VIEW already exists.
- `SBLD_OR_REPLACE_VIEW` – generated DDL creates new `VIEW` or replaces existing `VIEW`.
- `SBLD_VIEW_IF_NOT_EXISTS` – generated DDL creates new `VIEW` when executed only if the VIEW does not exist.

All arguments values are `VARCHAR/STRING`. Arguments must be called using argument names. Positional[1] argument lists are not supported, e.g.:

```
SELECT SBLD__OR_REPLACE_VIEW (REPO_VIEW_NAME => 'VW_CUSTOMER )
SBLD_VIEW function  returns a VARCHAR result.

SBLD_OR_REPLACE_VIEW(
        REPO_VIEW_NAME => < required >,
        NEW_VIEW_COMMENT => 'NULL', -- Generated VIEW comment
        TO_DB => 'NULL',            -- Destination database
        TO_SCHEMA => 'NULL',        -- Destination schema
        NEW_VIEW_NAME => 'NULL',    -- Generated VIEW name
        FROM_CLAUSE => 'NULL',      -- FROM
```

[1] Positional arguments may be used if no optional arguments are used. REPO_VIEW_NAME is position #1.

```
        POST_FROM_CLAUSE => 'NULL', -- Clauses following generated
FROM,WHERE, GROUP BY, etc.
        VIEW_CONTROL => 'NULL',     -- Snowflake VIEW options, SECURE, TEMP,
etc.
        LIST_CONTROL => 'NULL',     -- Controls generation of pre-SELECT
column list
        ROW_ACCESS => 'NULL',       -- Role Based Access Control
        OBJECT_TAG => 'NULL'        -- Tag on generated VIEW
    );
```

See the Appendix for more information.

CREATE VIEW Pre-SELECT Column List Functions

The following functions assign properties to existing View Columns. AltaSQL automatically keeps the pre-SELECT column list aligned with the `SELECT` clause.

SBLD_ALTER_VIEW_COLUMN_DATA_MASK

```
SBLD_ALTER_VIEW_COLUMN_DATA_MASK(
        REPO_VIEW_NAME => < required >,
        VIEW_COLUMN => < required >,
        DATA_MASK => < required >
    );
```

`DATA_MASK` is the specification of the Snowflake Dynamic Data Masking Policy. "`WITH MASKING POLICY`" and "MASKING POLICY" wording is optional; AltaSQL will auto correct the specification.

SBLD_ALTER_VIEW_COLUMN_TAG

```
SBLD_ALTER_VIEW_COLUMN_TAG(
        REPO_VIEW_NAME => < required >,
        VIEW_COLUMN => < required >,
        COLUMN_TAG => < required >
    );
```

`COLUMN_TAG` is the tag specification. "`WITH TAG`" and "`TAG`" wording is optional. AltaSQL will auto correct the specification.

SBLD_CTAS_FROM_VIEW

Creates a `TABLE` from a View Definition. It follows the same conventions as the Create View functions for defining the source view, i.e., `DEFAULT_SOURCE_DB and DEFAULT_SCHEMA_DB`. If a `FROM_CLAUSE` is used, it should be appropriately qualified.

```
FUNCTION SBLD_CTAS_FROM_VIEW (
      REPO_VIEW_NAME => <required>,
      TABLE_NAME => <required>,
      TO_DB VARCHAR => 'NULL',
      TO_SCHEMA VARCHAR => 'NULL',
      FROM_CLAUSE VARCHAR => 'NULL',
      POST_FROM_CLAUSE => DEFAULT 'NULL',
      TABLE_COMMENT VARCHAR => 'NULL',
      TABLE_CONTROL VARCHAR => 'NULL',
      DATA_RETENTION_TIME_IN_DAYS => NULL,
      ROW_ACCESS VARCHAR =>'NULL',
      OBJECT_TAG VARCHAR =>'NULL')
```

TABLE_CONTROL determines table type and CREATE option. `[`
`[TEMP | TEMPORARY | VOLATILE | TRANSIENT ]`
`[NOT_EXISTS |OR_REPLACE]`
`]`

Plug-n-Play for Modular Construction

Functions[2] with optional FROM_CLAUSE and POST_FROM_CLAUSE arguments enable the generation of SQL statements in a modular fashion. The SELECT clause may be generated dynamically using the values of the FROM_CLAUSE and POST_FROM_CLAUSEs.

The arguments values for these clauses may be:

- NULL or 'NULL'
    - If the FROM_CLAUSE is set to one of the above, the FROM clause is generated from the DEFAULT_SOURCE_DB, DEFAULT_SOURCE_SCHEMA and SOURCE_OBJECT values; the POST_FROM_CLAUSE may still be used.
- Strings
- Session variables
- `SBLD_GET_TAG_VALUE` function call embedded in $$<function_call>$$

WARNING: Some 3rd party SQL Editors[3], may have issues formatting ASDF function calls. In particular, session variables embedded in the $$ $$ construct. Contact the vendor for a fix or use true values instead of the session variable.

[2] Currently this includes the functions that generate SELECT, CREATE VIEW DDL, SELECT_DISTINCT and CTAS DDL.
[3] DBVisualizer formatting requires **Tools > Tool Properties** and go to **General > SQL Commander > Statement Delimiters** and check the **Allow SQL Dialects**.

New Expression Functions

AltaSQL Enterprise adds new functions to simplify adding common types of expressions.

SBLD_ADD_CONSTANT

```
SBLD_ADD_CONSTANT(
        REPO_VIEW_NAME => < required >,
        NEW_COLUMN_NAME => < required >,
        CONSTANT => < required >,
        COLUMN_ORDER => < required >,
        CAST_TO => 'NULL',
        CONSTANT_COMMENT => 'NULL'
    );
```

Adds: `<numeric> AS <new_column_name>` to generated SQL.

SBLD_ADD_LITERAL

```
SBLD_ADD_LITERAL(
        REPO_VIEW_NAME => < required >,
        NEW_COLUMN_NAME => < required >,
        LITERAL => < required >,
        COLUMN_ORDER => < required >,
        LITERAL_COMMENT => 'NULL'
    );
```

Adds: `<string> AS <new_column_name>`

Use $$ construct for embedded quotes.

SBLD_ADD_NULL_AS

```
SBLD_ADD_NULL_AS(
        REPO_VIEW_NAME => < required >,
        NEW_COLUMN_NAME => < required >,
        COLUMN_ORDER => < required >,
        NULL_AS_COMMENT => < required >
    );
```

Adds: `NULL AS <new_column_name>`

SBLD_ADD_UDF_CALL

```
SBLD_ADD_UDF_CALL(
        REPO_VIEW_NAME => < required >,
        NEW_COLUMN_NAME => < required >,
        UDF_CALL => < required >,
        COLUMN_ORDER => < required >,
        SOURCE_DB => 'NULL',
        SOURCE_SCHEMA => 'NULL',
        UDF_CALL_COMMENT => 'NULL'
    );
```

Adds: `<source_db>.<source_schema>.<udf_call> AS <new_column_name>`

`SOURCE_DB` and `SOURCE_SCHEMA` are optional.

SBLD_ALTER_VIEW_COLUMN_JSON_PATH

```
SBLD_ALTER_VIEW_COLUMN_JSON_PATH(
    REPO_VIEW_NAME => < required >,
    VIEW_COLUMN => < required >,
    JSON_PATH => < required >,
    CAST_TO => 'NULL'
);
```

Generates: `<view_column_name>:<json_path> AS <view_column_name>`
if `CAST_TO => 'NULL'`
from the existing semi-structured <view_column_name>.

Generates:
`<view_column_name>:<json_path>::<cast_to> AS <view_column_name>`
if `CAST_TO => <other>`
Alters an existing column. Copy the existing column to add as a new expression.

SBLD_ALTER_VIEW_COLUMN_CAST_TO

```
SBLD_ALTER_VIEW_COLUMN_CAST_TO(
        REPO_VIEW_NAME => < required >,
        VIEW_COLUMN => < required >,
        CAST_TO => < required >
    );
```

Adds: `CAST( <source_column> AS <target_data_type>) AS <view_column_name>`

If the `SOURCE_COLUMN_DATA_TYPE = JSON_PATH'` then it appends the data type to the JSON Path.

Common Table Expressions (CTEs)

Enterprise CTE support provides the addition of CTEs to generated SQL, providing the common capabilities of CTEs. CTEs may be added to View Definitions and generated in SELECT, CREATE VIEW and CTAS statements.

CTE Functions

SBLD_ADD_CTE

```
SBLD_ADD_CTE (
      REPO_VIEW_NAME => <required>,
      CTE_NAME => <required>,
      CTE_BODY => <required>,
      CTE_ORDER => <required>,,
      CTE_COMMENT => 'NULL);
```

Adds a CTE to a View Definition. AltaSQL automatically generates WITH followed by the CTEs. It will add parentheses if needed and strip a trialing semi-colon. This allows the use of

`CTE_BODY => SBLD_SELECT`

And

```
CTE_BODY => SBLD_GET_TAG_VALUE
```

And function calls in addition to hand crafted `CTE_BODY`.

Note that Snowflake ignores generated CTEs that do not have columns from the CTE in the SELECT clause, unless the CTE is included in the FROM clause. Thie allows incremental and modular development for View Definitions with multiple CTEs.

SBLD_GET_CTE

```
SBLD_GET_CTE(
      REPO_VIEW_NAME => <required>,
      CTE_NAME       => <required>)
```

Returns the CTE_BODY

SBLD_ALTER_CTE

```
SBLD_ALTER_CTE(
      REPO_VIEW_NAME  => <required>,
      CTE_NAME        => <required>,
      NEW_CTE_BODY    => <required>,
      NEW_CTE_ORDER   => 'NULL',
      NEW_CTE_COMMENT => 'NULL')
```

Alter the definition of a CTE

Using CTEs

Although AltaSQL generates CTEs, it does not automatically add CTE columns to the View Definition.

References to the CTE columns must be added as using `SBLD_ADD_VIEW_COLUMN` to the View Definition using the `CTE_NAME` as the `SOURCE_OBJECT`. `DEFAULT_SOURCE_DB` and `DEFAULT_SOURCE_SCHEMA` should be `NULL`/`'NULL'`. The other optional arguments may be specified. The `SOURCE_OBJECT` and `TABLE_ALIAS` should be consistent for all added columns form the CTE; different combinations may result in unexpected results.

```
SELECT
    ASQL_DB.ASDF_SCHEMA.SBLD_ADD_VIEW_COLUMN(
        REPO_VIEW_NAME => 'VW_TRACK_FACT',
        NEW_VIEW_COLUMN => 'Genre',
        SOURCE_COLUMN => 'Name',
        SOURCE_OBJECT => 'GENRE_DIM', -- CTE_NAME
        COLUMN_ORDER => 2,
        SOURCE_COLUMN_DATA_TYPE => 'NAME',
        DEFAULT_SOURCE_DB => 'NULL',
        DEFAULT_SOURCE_SCHEMA => 'NULL',
        TABLE_ALIAS => 'NULL',
        COLUMN_COMMENT => 'Genre Name from Genre Dimension',
        CAST_TO => 'NULL',
```

```
JSON_PATH => 'NULL',
COLUMN_TAG => 'NULL',
DATA_MASK => 'NULL );
```

An appropriate `FROM_CLAUSE` must be supplied for generating `SELECT, CREATE VIEW` and `CTAS` statements. To quickly build an appropriate `FROM clause`, generate a `SELECT` statement without specifying a `FROM_CLAUSE` argument. The generated SELECT statement will generate a template for a `FROM clause` listing all of the `SOURCE_OBJECTs`, e.g.,

-- ??Too Many SOURCE_OBJECTS - USE FROM_CLAUSE ?? "ALBUM_DIM"
-- ?? Too Many SOURCE_OBJECTS- Use FROM_CLAUSE ??"GENRE_DIM";

Modular Construction with AltaSQL Tags

This section describes an architecture for maximizing your use of AltaSQL for generating SQL statements. This is particularly useful when generating SELECT, CREATE VIEW DDL and CTAs statements.

The functions `SBLD_GET_TAG_VALUE` and `SBLD_GET_QUOTED_TAG_VALUE` may be used as values for virtually all AltaSQL Defined Functions. Both functions use the same arguments:

```
(REPO_VIEW_NAME => <required>
TAG => <required>
TAG_CLASS => 'NULL')
```

Predefined Tag Classes for Consistency and Governance

AltaSQL Tag Classes provide a framework for ensuring consistency and governance in the AltaSQL meta-data catalog.

To create and use AltaSQL tags, you must first define AltaSQL tag classes. AltaSQL has a set of suggested Built-in Tag Classes[4], or you can create your own.

Built-in Tag Classes may be found in the shared database table:
META_DATA_REPO_SCHEMA.ALTASQL_DEFINED_TAG_CLASSES[5].

`Tag Class Names are n`ot case sensitive. 'NULL" is always accepted as a Tag Class Name.

The TAG_CLASS column in the table is the Tag Class Name. The `DESCRIPTION` column is the suggested usage; AltaSQL does not verify that the `TAG_VALUE in a Tag` matches the DESCRIPTION.

[4] See *Built-in* Tag Classes for a list of built-in AltaSQL Tags as of 4/23/2025
[5] As we may add new Tag Classes to this table, we suggest reviewing the table before creating new Tag Classes.

You can create TAG_CLASSes with the same Tag Class Name as a built-in TAG_CLASS with a different TAG_CLASS_COMMENT. These are created as AltaSQL Tags in the designated View Definition.

User created TAG_CLASSes with user defined Tag Class Names persist as long as at least one such TAG_CLASS definition exists.

The primary use case for TAG_CLASSes is to treat AltaSQL Tags with the same Tag Name and different TAG_CLASS Name as attributes of an "object" within a View Definition. This provides a robust meta-data model for AltaSQL Tags.

We recommend creating a dedicated View Definition in which to create your own TAG_CLASSes.

AltaSQL Tag Classes must be defined before a Tag can be added with the exception of the 'NULL' Class. Tag Class names are not case sensitive. The following function is used to add AltaSQL Tag Classes:

```
SBLD_ADD_TAG_CLASS (
      REPO_VIEW_NAME => <required>
      TAG_CLASS => <required>,
      TAG_CLASS_COMMENT => 'NULL')
```

The arguments for SBLD_ADD_TAG are:

```
SBLD_ADD_TAG(
CREPO_VIEW_NAME => < required >,
VTAG => <required>,
        TAG_VALUE => 'NULL',
        TAG_CLASS => 'NULL',
        TAG_COMMENT => 'NULL',
        COPY_CONTROL => 'NULL')
```

Using Session Variables vs Tag Class Names

Using a set of predefined session variables in your working environment can provide a handy *reference* to true Class Tag name, e.g.,

```
SET FROM_CLAUSE = 'FROM_CLAUSE';
```

Although both the session variable and the value of the session variable may be used in function calls, we recommend using the actual Tag Class Name rather than the session variable in function calls for the following reasons:

- Using the Tag Class Name is portable; it does not require setting the session variable which may be problematic when executing in procedural code. This applies to session variables in general.

- Due to a Snowsight bug, formatting SQL statements in Snowsight with a session variable contained in a $$ $$ construct changes the semantics of the function call; the resulting SQL produces an incorrect result.

While session variables are very useful when working in worksheet, they should be avoided for function calls in procedural code.

SQL Statement Categories

We define the following categories of queries.

Dynamically Generated SQL

These statements are generated from AltaSQL functions, and the result must be executed by the user or procedure with appropriate ROLE and warehouse.

"Ready-to-Execute" Queries

Generated SQL statements may be added as an AltaSQL Tag using the recommended `TAG_CLASS $RTE_QUERY`.

These are typically complete `SELECT`, `CREATE VIEW` or `CTAS` statements. The user simply uses SBLD_GET_TAG_VALUE to retrieve and execute the statement. This is an ideal way to share SQL statements.

You can use this technique to bring your own existing statements into AltaSQL; any SQL statement may be used.

“Ready to Run ASDFs”

Function calls with supplied argument values may be saved as AltaSQL Tags. This enables execution from procedural code as well as editing the function call argument values for different results. Avoid the use of session variables in Ready to Run ASDFs.

Creating View Definitions with Multiple SOURCE_OBJECTs

Adding Existing Snowflake Objects to a View Definition

`SBLD_ADD_SOURCE_OBJECT_TO_VIEW` function generates an `INSERT` statement that creates a `SOURCE_OBJECT` and associated columns from *an existing View Definition*. Note that the Source Object may be *any* Snowflake object with *column names matching those of* *`FROM_REPO_VIEW_NAME`*. If the FROM_REPO_VIEW_NAME definition contains multiple SOURCE_OBJECTS, they are treated as existing Snowflake objects.

```
FUNCTION SBLD_ADD_SOURCE_OBJECT_TO_VIEW(
        TO_REPO_VIEW_NAME => < required >,
        FROM_REPO_VIEW_NAME => < required >,
        SOURCE_OBJECT_NAME => 'NULL',
        SOURCE_DB => 'NULL',
        SOURCE_SCHEMA => 'NULL',
        SOURCE_TABLE_ALIAS => 'NULL'
    );
```

The default for `SOURCE_OBJECT_NAME` is `FROM_REPO_VIEW_NAME`.

The default for `SOURCE_DB` is the value of `DEFAULT_SOURCE_DB` for the `FROM_REPO_VIEW_NAME` definition.

The default for `SOURCE_SCHEMA` is the value of `DEFAULT_SOURCE_SCHEMA` for the `FROM_REPO_VIEW_NAME` definition.

The default for `TABLE_ALIAS` is `NULL`. It is not copied from the `FROM_REPO_VIEW_NAME` definition.

It does not copy `TABLE_ALIAS`, `COLUMN_TAG`, `DATA_MASK`, `CAST_TO`, `JSON_PATH`, `SOURCE_COLUMN_DATA_TYPE`, `AltaSQL Tags` and `DELETED VIEW_COLUMNS` from `FROM_REPO_VIEW_NAME`

The result typically requires modification, in particular, duplicate column names for JOIN columns, e.g.,

```
SELECT SBLD_IGNORE_VIEW_COLUMN
      REPO_VIEW_NAME => <required>, VIEW_COLUMN => <required>);
```

Adding Source Objects from Existing View Definitions

```
FUNCTION SBLD_ADD_VIEW_COPY_FROM_REPO_VIEW(
      TO_REPO_VIEW_NAME  => <required>,
      FROM_REPO_VIEW_NAME  => < required >   )
```

Generate as `INSERT .. SELECT` statement copying columns from `FROM_REPO_VIEW_NAME`. All VIEW_COLUMN properties are copied following the rules for copying AltaSQL Tags.

TO_REPO_VIEW_NAME must be the name of an existing View Definition. Modify the generated INSERT's WHERE clause and other parts of the SELECT as appropriate.

Building View Definitions with Multiple Source Object (VMSO)

VMSOs require careful planning and consideration. In particular:

- Defining JOIN conditions
- Handling duplicate column names.

Defining JOIN conditions

AltaSQL does not generate a usable default FROM clause for a VMSO. It generate a list of qualified SOURCE_OBJECT which may be easily edited to create a usable FROM_CLAUSE,e.g.,

```
FROM
    "CHINOOK_DB"."CHINOOK_PHYSICAL_SCHEMA"."Customer" AS CUST
    -- ?? Too Many SOURCE_OBJECTS- Use FROM_CLAUSE ??
    "CHINOOK_DB"."CHINOOK_VIEW_SCHEMA"."VW_INVOICE_NL" AS INV
```

All of the JOIN conditions must be defined to form a valid FROM clause. This enables the creation of multiple FROMs clauses that can be used with Plug-n-Play.

You can store these as AltaSQL Tags and use FROM_CLAUSE => SBLD_GET_TAG_VALUE to generate SQL.

View Column Name Conflicts

Joining multiple source objects typically results in duplicate column name existing in multiple source objects, e.g., primary and foreign key columns.

Resolving these issues in handcrafted SQL scripts can be a frustrating and intensive effort.

AltaSQL simplifies identifying duplicate columns with the ASDF call:

```
SELECT * FROM TABLE(SBLD_LIST_DUPLICATE_VIEW_COLUMNS();
```

This returns a list of the duplicate View Columns with all of the columns from the meta-data repository. Specify meta-data columns and filters, e.g.,

```
SELECT VIEW_NAME,VIEW_COLUMN,SOURCE_OBJECT,DEFAULT_SOURCE_DB,DEFAULT_SOURCE_SCHEMA
FROM
    TABLE( SBLD_LIST_DUPLICATE_VIEW_COLUMNS() )
WHERE
    VIEW_COLUMN ILIKE 'ADDRESS'
AND SOURCE_OBJECT ILIKE '%CUST%';
```

Returns：

VIEW_NAME	VIEW_COLUMN	SOURCE_OBJECT	DEFAULT_SOURCE_DB	DEFAULT_SOURCE_SCHEMA
VW_CUSTOMER_BASE	Address	Customer	CHINOOK	CHINOOK_PHYSICAL_SCHEMA
VW_CUSTOMER_COPY	Address	Customer	CHINOOK	CHINOOK_PHYSICAL_SCHEMA

AltaSQL's use of meta-data and column-alias syntax in generated SQL simplifies this process by:

- Setting View Columns to "IGNORE" inhibiting generating columns while preserving important meta-data for use in JOIN conditions
    - `SBLD_ALTER_VIEW_IGNORE_COLUMN`
- Renaming duplicated columns
    - `SBLD_ALTER_VIEW_RENAME_COLUMN`
- Deleting unwanted columns
    - `SBLD_ALTER_VIEW_DELETE_VIEW_COLUMN`

AltaSQL can address these issues via multiple methods; the following are suggestions and recommendations:

- Modifying individual View Columns
- Using bulk techniques to generate scripts with individual ASDF calls
- Modifying DML generated by ASDFs
- Spreadsheets

There are two general workflows for resolving View Column conflicts:

- Before repository modification when number of columns is large; consider
    - Spreadsheet
    - Create working repository or working View Definition prior to adding source object
- After repository modification for small number of columns

Which workflow to use depends on:

1. How the repository is used:
    a. Is it closely managed and governed?
        i. Test/stage in working repository prior to loading/update repository
    b. Is it a shared within a small team?
        i. Set standards, ideally with a single repository owner
        ii. ASDF calls
    c. Is it a "power user" repository?
        i. Power user owns and manage repository
2. Your naming and governance conventions
    a. For large number of involved View Columns, spreadsheet are recommended
    b. For experienced SQL users, consider modifying AltaSQL generated meta-data DML
3. The number of source objects and duplicated columns

Appendices

Available Documentation

Additional documentation is available on our website Documentation page. As documentation is updated as needed, direct links are not practical. Additional relevant documentation:

- RMMS Technical Reference – includes detailed ASDF descriptions for all AltaSQL functions
- AltaSQL Admin and Governance
- AltaSQL Common User Guide
- AltaSQL SELECT Discover Quick Start

Detailed CREATE View Description

The FROM_CLAUSE is always generated based on the View Definition. CTEs may be used to provide additional functionality.

Argument Name	Data Type	Default if not specified	Result
REPO_VIEW_NAME	VARCHAR	Must be specified, else error	The VIEW_NAME in the meta-data used to generate the CREATE VIEW statement.
TO_DB	VARCHAR	If not provided, no DATABASE qualification is generated.	Qualifies DATABASE destination for new VIEW.
TO_SCHEMA	VARCHAR	If not provided, no SCHEMA qualification is generated.	Qualifies SCHEMA destination for new VIEW.
NEW_VIEW_NAME	VARCHAR	REPO_VIEW_NAME	The name of the generated object.
NEW_VIEW_COMMENT	VARCHAR	COMMENT on VIEW	Comment on the generated object.
FROM_CLAUSE	VARCHAR	Ignored	Not supported in Enterprise. Generate and error if use with a non-null value.
POST_FROM_CLAUSE	VARCHAR	Ignored	Additional SQL clauses (WHERE, ORDER BY, GROUP BY, etc.).

Argument Name	Data Type	Default if not specified	Result
			Joins with CTEs may be provided as POST_FROM_CLAUSE
ROW_ACCESS	VARCHAR	Ignored	ROW ACCESS POLICY
OBJECT_TAG	VARCHAR	Ignored	Snowflake OBJECT_TAG

Argument Name	Delimited string of one or more of following sub-arguments	Standard Snowflake CREATE VIEW options	
VIEW_CONTROL			
	SECURE	SECURE	
	TEMP	TEMP	
	TEMPORARY	TEMPORARY	
	VOLATILE	Being deprecated	
	GLOBAL	GLOBAL	
	RECURSIVE	RECURSIVE	
	COPY_GRANTS	COPY_GRANTS	

LIST CONTROL	Delimited string of one or more of following sub-arguments	Column list result	Default if not specified
	NO_COLUMN_COMMENTS	Do not generate COLUMN COMMENT	Generated column COMMENTs.
	NO_COLUMN_TAGS	Do not generate COLUMN TAG	Generate column Snowflake tags
	NO_DATA_MASKS	Do not generate DYNAMIC DATA MASKs	Generate dynamic data masks
	FORCE_LIST	Always generate Column list	Generate Column List only if SBLD_NOT_NULL

		using prior options.	for any of [COMMENTs, TAGs or DATA_MASKs] exist.
	NO_LIST	Do not generate column list	Do not generate column list.

Argument Results

1. Enterprise supports only a single SOURCE_OBJECT for the generated FROM clause.
2. If a POST_FROM_CLAUSE argument is provided, it will be appended. This Includes WHERE, GROUP BY, ORDER BY, etc. Joins with the generated FROM clause and CTEs may be specified in the POST_FROM_CLAUSE.
3. If NEW_VIEW_NAME argument is ***not*** provided, the value of REPO_VIEW_NAME is the name of the generated VIEW DDL.
4. TO_SCHEMA is used to qualify the schema of the generated object. If not provided or set to an SBLD_IS_NULL value, the DEFAULT_SOURCE_SCHEMA value is used.
5. TO_DB is used to qualify the database of the generated object. If not provided, the value of DEFAULT_SOURCE_DB is used. The default can be overridden by setting TO_DB => VARCHAR DEFAULT 'NULL'.
6. NEW_OBJECT_COMMENT generated the appropriate COMMENT in the generated DDL. Setting NEW_OBJECT_COMMENT => VARCHAR DEFAULT 'NULL' results in no COMMENT.

Built-in Tag Classes

Following is a list of Tag Classes as of this the time of this publication; we add additional Tag Classes as we uncover more use cases. This has no effect on user defined Tag Classes.

The table, *META_DATA_REPO_SCHEMA"."ALTASQL_DEFINED_TAG_CLASSES,* in the shared database contains the latest list.

TAG_CLASS	DESCRIPTION
CODE_BASE	Base logic or foundational reusable code components
COLUMN_TAG	Snowflake Column Tag
CREATE_VIEW_HDR	CREATE VIEW header metadata, such as comments or options
CTAS_HDR	Header info or template for CTAS (Create Table As Select) generation
CTE	Common Table Expression definitions (CTE wrappers)
CTE_BODY	Core logic or body of a CTE statement
DATA_MASK	Data masking policy definitions
EXPRESSION	SQL expressions (reusable in columns or clauses)
FROM_CLAUSE	FROM clause definitions
OBJECT_TAG	Snowflake Object Tag
PATH	A JSON Path
POST_FROM_CLAUSE	WHERE, GROUP BY, ORDER BY, etc. clauses following FROM
RBAC	Role-Based Access Control metadata or guidance
RUN_ASDF	Ready to Generate ASDFs (AltaSQL function calls ready to be executed)
RUN_READY	Ready to Execute SQL queries (complete SQL statements)
SBLDR_META_REPOSITORY	Identifies or manages repository-level meta-data structures
SELECT_CLAUSE	SELECT clause templates or fragments
SOURCE_INFO	Information about the origin of a View Definition or data lineage
SUBQUERY	A SQL subquery
TAG_CLASS	User created AltaSQL Tag defining an AltaSQL Tag Class
TAG_INFO	Information about an AltaSQL Tag
TAG_NAME	Tag Names
TAG_VALUE	Tag Values
UDF_CALL	UDF Call
USAGE	Notes or usage instructions for a View Definition or component
VIEW_INFO	Information about the View Definition