SQreamDB

Release 4.12

SQreamDB Documentation

Apr 01, 2026

CONTENTS:

1 AISQream Documentation **3**

1.1 About SQream . . . 3
1.2 Hardware requirements . . . 3
1.3 Embedded ML Models . . . 3
1.3.1 Linear Regression . . . 4
1.3.1.1 Syntax . . . 4
1.3.1.2 Parameters . . . 4
1.3.1.3 Usage Notes & Limitations . . . 5
1.3.2 XGBoost . . . 5
1.3.2.1 Parameters . . . 6
1.3.2.1.1 Global Configuration . . . 6
1.3.2.1.2 General Parameters . . . 6
1.3.2.1.3 Booster Parameters (applicable when booster is set to gbtree or dart) . . . 7
1.3.2.1.4 Additional parameters for Dart Booster . . . 8
1.3.2.1.5 Parameters for Linear Booster . . . 8
1.3.2.1.6 Learning Task Parameters . . . 9
1.3.2.1.7 Parameters for Tweedie Regression . . . 9
1.3.2.1.8 Parameter for using Pseudo-Huber . . . 10
1.3.2.1.9 Parameter for using Quantile Loss . . . 10
1.3.2.1.10 Parameter for using AFT Survival Loss survival: AFT and Negative Log Likelihood of AFT metric . . . 10
1.3.2.1.11 Parameters for learning to rank . . . 10
1.3.2.1.12 Command Line Parameters . . . 10
1.3.2.2 Usage Notes & Limitations . . . 11
1.4 Model Training via Python . . . 11
1.4.1 Syntax . . . 11
1.4.2 Usage notes . . . 12
1.5 Export Model . . . 12
1.5.1 Syntax . . . 12
1.5.2 Parameters . . . 12
1.5.3 Example . . . 13
1.6 Validation Functions . . . 13
1.6.1 How to Use Model Validation Functions . . . 13
1.6.2 MSE / RMSE (for Regression) . . . 13
1.6.3 Syntax . . . 13
1.6.4 Parameters . . . 13
1.6.5 Example: . . . 13
1.6.6 Area under curve (for XGBoost) . . . 14
1.6.7 Syntax . . . 14
1.6.8 Parameters . . . 14

1.6.9 Example: . . . 14
1.6.10 Usage notes . . . 14
1.7 Python modules - Installation & Configuration . . . 14
1.7.1 Configurations: . . . 15
1.7.2 User notes & limitations: . . . 16
1.7.2.1 Environmental: . . . 16
1.7.2.2 Python module execution notes: . . . 16
1.7.2.3 Unsupported functionalities: . . . 16
1.7.3 Logs: . . . 16
1.8 Python Functions . . . 19
1.8.1 1. Syntax Overview . . . 19
1.8.1.1 **Module Creation:** . . . 19
1.8.1.2 **Python Table Functions (PTFs): The <function_clause>** . . . 20
1.8.1.3 **Python Scalar Functions (PSFs): The <scalar_function>** . . . 20
1.8.1.4 **Example: Defining a Module with Both Function Types** . . . 21
1.8.2 2. Usage Examples . . . 21
1.8.2.1 Example 1: Python Scalar Function (PSF) . . . 21
1.8.2.2 Example 2: Passing Literal Parameters For Python Scalar Function (PSF) . . . 22
1.8.2.3 Example 3: Python Table Function (PTF) . . . 23
1.8.2.4 Example 4: Passing Literal Parameters For Python Table Function (PTF) . . . 24
1.8.2.5 Example 5: Join Statment on a Python Table Function (PTF) . . . 24
1.8.3 3. Return Values . . . 26
1.9 Catalog Tables . . . 26
1.9.1 Catalog Table: ml_models . . . 26
1.9.2 Catalog table: ml_linear_reg . . . 27
1.9.3 Catalog table: ml_xgboost . . . 28
1.9.4 Catalog table: py_modules . . . 29
1.9.5 Catalog table: py_module_permissions . . . 29
1.9.6 Catalog table: registered_algorithms . . . 30
1.10 Security Requirements . . . 30
1.11 SQream Integration with Apache Iceberg . . . 30
1.11.1 Overview of Apache Iceberg Integration . . . 31
1.11.1.1 Iceberg Architecture: . . . 31
1.11.2 Connectivity to Iceberg . . . 31
1.11.3 DDL Operations on an Iceberg Table . . . 32
1.11.3.1 Data Type Mapping . . . 34
1.11.4 Querying an Iceberg Table . . . 34
1.11.4.1 Time Travel . . . 34
1.11.4.2 Extended Metadata Queries . . . 35
1.11.5 Write operations on an Iceberg Table . . . 37
1.12 Access Control . . . 38
1.12.1 Permissions . . . 38
1.12.1.1 Objects . . . 38
1.12.1.2 Syntax . . . 38
1.12.1.2.1 GRANT . . . 39
1.12.1.2.2 REVOKE . . . 39
1.12.1.3 Examples . . . 40
1.12.1.3.1 GRANT . . . 40
1.12.1.3.2 REVOKE . . . 41

This is an **alpha version** of AISQream. Expect limitations and ongoing development.

CHAPTER

ONE

AISQREAM DOCUMENTATION

This is an **alpha version** of AISQream. Expect limitations and ongoing development.

1.1 About SQream

SQream is A data analytics company that helps organizations break through barriers to ask the biggest, most important questions from their data. Our GPU-based technology empowers businesses to overcome dataset limits and query complexity to analyze exponentially more data and get substantially faster insights at dramatic cost-savings. By leveraging SQream's advanced analytics capabilities for AI/ML, enterprises can stay ahead of their competitors while reducing hardware usage. SQream fuels AI Factories by providing GPU-accelerated software to streamline massive data processing and ML development, enabling faster insights and scalable AI at every stage. If you want to take your data initiatives to the next level, Ask Bigger and unlock new opportunities with SQream.

To learn more, visit https://sqream.com or follow us on Twitter, LinkedIn, or Facebook

1.2 Hardware requirements

Hardware requirements for AISQream are GPUs that support CUDA 12.x or higher and have a Volta or newer architecture. The link you provided lists the GPUs that support vGPU, but it's not a direct list of Volta or newer architectures. You can find more information about NVIDIA's GPU architectures on their website.

https://docs.nvidia.com/vgpu/gpus-supported-by-vgpu.html

1.3 Embedded ML Models

SQream offers support for embedded and Python-based ML models. AISQream empowers users to build and train machine learning models, including Linear Regression and XGBoost, directly within the database using SQL.

- *Linear Regression*
- *XGBoost*

1.3.1 Linear Regression

1.3.1.1 Syntax

```
--# Create and train model
CREATE [OR REPLACE] MODEL [database.schema.]model_name
OPTIONS(model_option_list)
AS {query_statement};
```

```
model_option_list:
  MODEL_TYPE = { 'LINEAR_REG' | 'XGBOOST' }
, MODEL_PATH = 'PATH/TO/MODEL/JSON/TO/CREATE/WITH_NAME_OF_MODEL_INCLUDED.json'
[, INITIAL_ALGORITHM = { 'SVD' | 'EIGENDECOMPOSITION' | 'QR_DECOMPOSITION' } ]
[, GD_OPTIMIZER = { 'SGD' | 'ADAM' } ]
[, STANDARDIZATION = TRUE | FALSE ]
[, LEARNING_RATE = FLOAT ]
[, EPOCH_COUNT = INT ]
[, LOSS_FUNCTION = { 'Mse', 'L1', 'SmoothL1', 'Huber' } ]
[, TOLERANCE = FLOAT ]
```

```
--# Inference
SELECT model_predict(
    [database.schema.]model_name,
    feature_col1 [,feature_column2, ...])
FROM {query_statement};

--# DROP
DROP MODEL [database.schema.]model_name;
```

1.3.1.2 Parameters

Table 1: Linear Regression Parameters

Parameter Name	Description	Default Value
MODEL_TYPE	Specify the algorithm.	Mandatory
MODEL_PATH	Specify the path to save the model to.	Mandatory
INITIAL_ALGORITHM	Algorithm to fit linear model.	SVD
GD_OPTIMIZER	Optimizer used in second phase (gradient descent).	ADAM
STANDARDIZATION	Normalize features (mean = 0, std = 1).	FALSE
LEARNING_RATE	Gradient descent step size.	0.01
EPOCH_COUNT	Number of full passes over data.	Optional
LOSS_FUNCTION	Distance between real and predicted values.	Mse
TOLERANCE	Minimal loss reduction needed to continue training.	0.01

1.3.1.3 Usage Notes & Limitations

- Based on Nvidia RAPIDS Linear Regression.
- Training and inference can be read directly from a table or a query expression.
- At least 2 columns should be provided for training (feature column and a label).
- The label column is the last column in the chunk's input for training.
- The model will be saved under database.schema hierarchy, like any other SQream object.
- model_predict doesn't work within sub-query.
- Model export functionality is in development.
- Up to 8K features limit is recommended in this current version.

1.3.2 XGBoost

Create and train model

```
CREATE [OR REPLACE] MODEL [database.schema.]model_name
OPTIONS(model_option_list)
AS {query_statement};
```

`model_option_list`:

```
MODEL_TYPE = 'XGBOOST',

MODEL_PATH = 'PATH/TO/MODEL/JSON/TO/CREATE/WITH_NAME_OF_MODEL_INCLUDED.json' -- must␣
↪for xgboost

[VERBOSITY] = {silent | info | warning | debug}

[BOOSTER] = { 'GBTREE' | 'GBLINEAR' | 'DART' }

[,treeBoosterParams]

[,dartParams]

[,linearBoosterParams]
```

`treeBoosterParams`: (applicable when BOOSTER = { 'GBTREE' | 'DART' }`)

```
[,ETA | LEARNING_RATE] : float
[,GAMMA | MIN_SPLIT_LOSS]
[...]
-- please view the parameters table under Booster Parameters section
```

`dartParams`:

```
[,SAMPLE_TYPE] = { 'UNIFORM' | 'WEIGHTED' }
[...]
-- please view the parameters table under Additional parameters for Dart Booster␣
↪section
```

`linearBoosterParams`:

```
[UPDATER] = { 'SHOTGUN' | 'COORD_DESCENT' }
[...]
-- please view the parameters table under Additional parameters for Linear Booster␣
↪section
```

Inference

```
SELECT model_predict(
[database.schema.]model_name,
feature_col1 [,feature_column2, ...])
FROM {query_statement};
```

DROP

```
DROP MODEL [database.schema.]model_name;
```

```
CREATE MODEL mod
OPTIONS
( model_type = 'xgboost'
, path='/home/sqream/mod.json' --full path
, max_depth = 10, eta = 0.3, max_bin=100
, verbosity= 'SILENT', tree_method='APPROX'
) AS
SELECT * from t_train;
```

1.3.2.1 Parameters

XGBoost Parameters — XGBoost 2.1.3 documentation. The parameters can be divided into global parameters (appear in the syntax), general parameters (available per booster) and learning task parameters.

1.3.2.1.1 Global Configuration

Parameter Name	Values	Comments
verbosity	“silent, info, warning, debug”	“Optional, default is silent”

1.3.2.1.2 General Parameters

Parameter Name	Values	Comments
booster	“gbtree, gblinear or dart”	“gbtree and dart use tree based models while gblinear uses linear functions. Default is gbtree.”
disable_default_eval_met	boolean	“false by default.”

1.3.2.1.3 Booster Parameters (applicable when booster is set to gbtree or dart)

Parameter Name	Values	Comments
"eta, learning_rate"	"float [0, 1]"	"Step size shrinkage used in update to prevent overfitting. Optional, default = 0.3."
"gamma, min_split_lo	"float [0, MAX_FLOA	"Minimum loss reduction required to make a further partition on a leaf node of the tree. Optional, default = 0."
max_depth	"int [0, MAX_INT]"	"Maximum depth of a tree. Optional, default = 6."
min_child_w	"int [0, MAX_INT]"	"Minimum sum of instance weight (hessian) needed in a child. Optional, default = 1."
max_delta_st	"int [0, MAX_INT]"	"Maximum delta step we allow each leaf output to be. Optional, default = 0. If the value is set to 0, it means there is no constraint."
subsample	"float (0,1]"	"Subsample ratio of the training instances. Optional, default = 1. Setting it to 0.5 means that XGBoost would randomly sample half of the training data prior to growing trees."
sampling_metho	"uniform, gradient_based"	"Optional, default= uniform."
colsample_bytree	"float (0, 1]"	"This is a family of parameters for subsampling of columns. All colsample_by* parameters have a range of (0, 1] specify the fraction of columns to be subsampled. Optional, default = 1."
colsample_bylevel	"float (0, 1]"	
colsample_bynode	"float (0, 1]"	
"lambda, reg_lambda"	"float [0, MAX_FLOA	"L1 regularization term on weights. Increasing this value will make the model more conservative. Optional, default = 1."
"alpha, reg_alpha"	"float [0, MAX_FLOA	"L2 regularization term on weights. Increasing this value will make the model more conservative. Optional, default = 1."
tree_method	"auto, exact, approx, hist"	"The tree construction algorithm used in XGBoost. See description in the reference paper and Tree Methods. Optional, default = auto."
scale_pos_w	float	"Control the balance of positive and negative weights. Optional, default = 1."
refresh_leaf	boolean	"This is a parameter of the refresh updater. When this flag is True, tree leafs as well as tree nodes' stats are updated. When it is False, only node stats are updated. Optional, default = True."
process_type	"default, update"	"A type of boosting process to run. Optional, default = default. default: The normal boosting process which creates new trees. update: Starts from an existing model and only updates its trees."
grow_policy	"depthwise, lossguide"	"Controls a way new nodes are added to the tree. Currently supported only if tree_method is set to hist or approx. Optional, default= depthwise."
max_leaves	int	"Maximum number of nodes to be added. Not used by exact tree method. Optional, default = 0."
max_bin	int	"Only used if tree_method is set to hist or approx. Maximum number of discrete bins to bucket continuous features. Optional, default = 256."
num_parallel	int	"Number of parallel trees constructed during each iteration. This option is used to support boosted random forest. Optional, default = 1."

1.3.2.1.4 Additional parameters for Dart Booster

Parameter Name	Values	Comments
sample_type	"uniform, weighted"	"Type of sampling algorithm. uniform: dropped trees are selected uniformly. weighted: dropped trees are selected in proportion to weight. Optional, default = uniform."
normalize_type	"tree, forest"	"Type of normalization algorithm. Optional, default = tree."
rate_drop	float	"Dropout rate. Optional, default = 0.0."
one_drop	bool	"When this flag is enabled, at least one tree is always dropped during the dropout. Optional, default = false."
skip_drop	float	"Probability of skipping the dropout procedure during a boosting iteration. Optional, default = 0.0."

1.3.2.1.5 Parameters for Linear Booster

Parameter Name	Values	Comments
"lambda, reg_lambda"	"float [0, MAX_FLOAT]"	"L2 regularization term on weights. Increasing this value will make the model more conservative. Optional, default = 0."
"alpha, reg_alpha"	"float [0, MAX_FLOAT]"	"L1 regularization term on weights. Increasing this value will make the model more conservative. Optional, default = 0."
updater	"shotgun, coord_descent"	"Optional, default= shotgun."
feature_selector	"cyclic, shuffle, random, greedy"	"Feature selection and ordering method. Optional, default = cyclic."
top_k	int	"The number of top features to select in greedy and thrifty feature selector. The value of 0 means using all the features. Optional, default = 0."

1.3.2.1.6 Learning Task Parameters

Parameter Name	Values	Comments
objective	"reg_squarederror, reg_squaredlogerror, reg_logistic, reg_pseudohubererror, reg_absoluteerror, reg_quantileerror, binary_logistic, binary_logitraw, binary_hinge, count_poisson, survival_cox, survival_aft, rank_ndcg, rank_map, rank_pairwise, reg_gamma, reg_tweedie"	"Optional, default = reg_squarederror."
base_	float	"The initial prediction score of all instances, global bias. If base_margin is supplied, base_score will not be added. Default is not passing any params."
eval_r	"reg_squarederror, reg_squaredlogerror, reg_logistic, reg_pseudohubererror, reg_absoluteerror, reg_quantileerror, binary_logistic, binary_logitraw, binary_hinge, count_poisson, survival_cox, survival_aft, rank_ndcg, rank_map, rank_pairwise, reg_gamma, reg_tweedie, rmse (root mean square error)"	"Optional, default according to objective: rmsle default metric of reg_squaredlogerror, mphe default metric of reg_pseudohubererror objective."
seed		"Random number seed. Optional, default = 0."
seed_	boolean	"Seed PRNG deterministically via iterator number. Optional, default = false."

1.3.2.1.7 Parameters for Tweedie Regression

Parameter Name	Values	Comments
tweedie_variance_pow	"float (1,2)"	"Parameter that controls the variance of the Tweedie distribution range. Optional, default=1.5."

1.3.2.1.8 Parameter for using Pseudo-Huber

Parameter Name	Values	Comments
huber_slope	float	"A parameter used for Pseudo-Huber loss to define the δ term. Optional, default = 1.0."

1.3.2.1.9 Parameter for using Quantile Loss

Parameter Name	Values	Comments
quantile_alpha	"float [0, 1]"	"A scalar or a list of targeted quantiles. Optional, default = 0.5."

1.3.2.1.10 Parameter for using AFT Survival Loss survival: AFT and Negative Log Likelihood of AFT metric

Parameter Name	Values	Comments
aft_loss_distribution	"normal, logistic or extreme"	Optional

1.3.2.1.11 Parameters for learning to rank

Parameter Name	Values	Comments
lambdarank_pair_method	"mean, topk"	"Optional, default = topk."
lambdarank_num_pair_per	"int [1, MAX_INT]"	"It specifies the number of pairs sampled for each document when pair method is mean. Optional."
lambdarank_normalization		"Specify whether we need to debias input click data. Optional, default = true."
lambdarank_bias_norm	float	"Lp normalization for position debiasing, default is L2. Only relevant when lambdarank_unbiased is set to true. Optional, default = 2.0."
ndcg_exp_gain		"Whether we should use the exponential gain function for NDCG. Optional, default = true."

1.3.2.1.12 Command Line Parameters

Parameter Name	Values	Comments
epoch_count	int	"The number of rounds for boosting. Optional, default 10."

1.3.2.2 Usage Notes & Limitations

- Based on DMLC's XGBoost.
- Training and inference can be read directly from a table or a query expression.
- At least 2 columns should be provided for training (feature column and a label).
- The label column is the last column in the chunk's input for training.
- The model will be saved under database.schema hierarchy, like any other SQream object.
- *model_predict* doesn't work within sub-query.
- Model export functionality is in development.
- Input feature types are Nullable Float.
- Support only single label at this stage, label should appear last.
- Up to 8K features limit is recommended in this current version.

1.4 Model Training via Python

AISQream's Model Training via Python modules extends its machine learning capabilities beyond the currently embedded Linear Regression and XGBoost algorithms. This new feature allows for the seamless integration and utilization of a broader range of Python-based ML algorithms directly within SQream's GPU-accelerated environment, significantly reducing development time for new models.

1.4.1 Syntax

– Algorithm registration

```
REGISTER ALGORITHM 'algorithm_name'
OPTIONS
  ( alg_path = 'path_to_alg_file'
  [, train_method := Text -- defaults to 'train'
  , predict_method := Text -- defaults to 'predict'
  ]);
```

– Create and train model

```
CREATE [OR REPLACE] MODEL [database.schema.]model_name
OPTIONS(model_option_list)
AS {query_statement};
```

model_option_list:

```
MODEL_PATH := Text     -- path to save the model to
MODEL_TYPE := Text     -- algorithm name as registered
MODEL_PARAMS := Text   -- MODEL_PARAMS map of Strings
```

– Inference

```
SELECT model_predict(
    [database.schema.]model_name,
    feature_col1 [,feature_column2, ...])
FROM {query_statement}; --either <table_name> or (select_query) in parentheses
```

– Drop model

```
DROP MODEL [database.schema.]model_name;
```

– Drop Algorithm registration

```
UNREGISTER ALGORITHM 'algorithm_name';
```

1.4.2 Usage notes

- The label column is the last column in the chunk's input for training.
- Distinguish between `alg_path` in the `REGISTER ALGORITHM` command and `model_path` in the `CREATE MODEL` command: the former specifies the path to the Python script while the latter specifies the path to the model output.
- The python file referred to in `alg_path` should contain a function for training and a function for inference, both of which should accept the input data in `cudf.DataFrame` form, and a map of string parameters. The `model_path` parameter in the train function will contain the path the model should be saved to, and later extracted in the predict function in the form of `train_result_path`.
- Once registered, algorithms are persistent until removed by `UNREGISTER ALGORITHM` command.
- Python version is compatible with SQream's prerequisites 3.11
- Embedded algorithms can not be unregistered.
- `train_method` and `predict_method` indicate the function names in the Python file, default values are `train` and `predict` respectively.
- Parameters for Train will be passed by the user and parsed at runtime by the Python code.

1.5 Export Model

The Export Model functionality allows users to save trained machine learning models (Currently supported for Linear Regression and XGBoost only) to external files in specified formats like JSON or Pickle.

1.5.1 Syntax

```
EXPORT MODEL [[<database name>.]<schema name>.]<model_name> TO <path>
OPTIONS(output_type = {JSON | PICKLE});
```

1.5.2 Parameters

- **Output type:**
  - Linear Regression model - JSON & PICKLE
  - XGBoost model - JSON only

1.5.3 Example

```
CREATE OR REPLACE MODEL my_lr_model
OPTIONS ( model_type = 'Linear_reg' , learning_rate = 0.01 )
AS SELECT * FROM t;

EXPORT MODEL my_lr_model to '/tmp/b.pkl'
OPTIONS(output_type = PICKLE);
```

1.6 Validation Functions

Validation functions provide tools to evaluate the performance of trained machine learning models against test datasets, offering metrics like Mean Square Error (MSE), Root Mean Square Error (RMSE) and Area Under the Curve (AUC).

1.6.1 How to Use Model Validation Functions

Use these functions in a SELECT statement to evaluate a model's performance against a test dataset.

1.6.2 MSE / RMSE (for Regression)

Calculates the Root Mean Square Error for Linear Regression models.

1.6.3 Syntax

```
SELECT validation_functions.rmse(model_name, features, label) FROM test_table;
```

1.6.4 Parameters

- `'model_name'`: The name of your trained Linear Regression model.
- `features`: The column(s) the model uses for prediction.
- `labels`: The column with the true numeric values.

1.6.5 Example:

```
SELECT
    validation_functions.rmse(
        house_price_model,
        (bedrooms, square_feet),
        actual_price
    )
FROM
    housing_test_data;
```

1.6.6 Area under curve (for XGBoost)

Calculates the Area Under the Curve (AUC) for XGBoost binary classification models.

1.6.7 Syntax

```
SELECT validation_functions.area_under_curve(model_name, features, labels)
FROM test_table;
```

1.6.8 Parameters

- `'model_name'`: The name of your trained XGBoost model.
- `features`: The column(s) the model uses for prediction.
- `labels`: The column with the true binary labels (0 or 1).

1.6.9 Example:

```
SELECT
    validation_functions.area_under_curve(
        customer_churn_model,
        account_age, monthly_spend,
        has_churned
    )
FROM customer_test_data;
```

1.6.10 Usage notes

- These functions, like `SUM()` or `AVG()`, return a single value for the entire dataset and cannot be used with a `GROUP BY` clause.
- Feature and label columns must be numeric; rows containing `NULL` values will be ignored.
- When selecting a model type, use `rmse` for Linear Regression and `area_under_curve` for XGBoost.
- Using the incorrect function for a model type will result in an error.

1.7 Python modules - Installation & Configuration

SQream's Python Module enables users to integrate custom Python code and functions directly. This allows for advanced data manipulation and custom machine learning operations, all accelerated by GPU.

- *Configurations*
- *How to install & run Python Server*
- *User notes & limitations*
- *Logs*

1.7.1 Configurations:

Sqream configuration: In this version, Python module service is per Sqream worker only. It is required to set configuration change regarding communication details in Sqream config:

Config file	Flag name	Flag type	Default value
Sqream config (not legacy)	pythonModulesGrpcPort	Worker	50051
Sqream config (not legacy)	grpcGpuAllocatorPort	Worker	50052

Python module configuration: Service's configuration is located in etc/python_service_config.json:

```
{
    "port": 50051,
    "gpu_alloc_port": 50052
}
```

Note: Worker flag means it can't be changed while the worker is up, the change will occur only after Sqream workers restart. Python module flags must match Sqream worker's flags mentioned above.

How to install python module service

One-time installation

1. Get the latest version of python module service & extract the package:

```
tar -xvf <PYTHON_MODULE_SERVICE_PACKAGE>;
cd python-module-service;
```

2. Create Python3.11 virtual environment:

```
python3.11 -m venv my_venv
source my_venv/bin/activate
```

3. Install required Python3.11 libraries and run requirements.txt installation (this may take a few minutes):

```
sudo yum install -y python3.11-devel
pip3.11 install -r requirements.txt
```

Run Python module service

1. Activate virtual environment:

```
cd python-module-service;
source my_venv/bin/activate
```

2. Run Python module service:

```
python3.11 py_modules.py
```

1.7.2 User notes & limitations:

1.7.2.1 Environmental:

- Python version is limited to the SQream prerequisites compiled version, that means the user must align to the recent SQream version, and upgrading a python version, requires upgrading SQream package.
- Python code would run with default Linux privileges therefore could be potentially dangerous and need to be handled with caution.
- Python code runs using the privileges of the user account that the SQream worker process operates under.
- As mentioned above - In this version there can be a python module service per worker.

1.7.2.2 Python module execution notes:

- Python module's main purpose is for batch processing, which means, python functions will occur chunk by chunk separately. For example, for 'max' function (which is aggregational function), instead of getting one maximum value from all chunks, we will get the maximum per chunk.
- Chunk processing is limited to default chunk size (E.g. 1M rows), and cannot be customized when invoked.
- In addition to the defined memory limits, the Python module's internal RAM usage cannot be controlled, which may lead to out-of-memory (OOM) runtime errors. These memory constraints apply to all RAM consumed by the custom code executed within the Python Server.
- Python functions that would print to stdout would be visible only where Python module's process is running.
- Error handling should work properly, Python errors would get raised in Sqream as runtime errors.
- Python module file path — Only local paths are currently supported. "Local paths" refer to paths that are either relative or absolute within the Python Server's execution directory.

1.7.2.3 Unsupported functionalities:

- Nested Python module calls are not supported. In other words, a Python UDF cannot invoke another Python UDF or any additional function executed through the Python Server.
- The functionality is currently limited to the active database and does not support cross-database access.

1.7.3 Logs:

Python module service has a log configuration file. Logs can be either shown to the console, and also be exported to file (same as we have in Sqream's log4cxx log configuration). File path: `etc/python_service_log_properties` - This path is relative to the Python module service directory. File content:

Configuration file	Explanation
`[loggers]`	keys=root
`[handlers]`	keys=consoleHandler,fileHandler
`[formatters]`	keys=standardFormatter
`[logger_root]`	level=DEBUG, handlers=consoleHandler,fileHandler
`[handler_consoleHandler`	class=StreamHandler, level=INFO, formatter=standardFormatter, args=(sys.stdout,)
`[handler_fileHandler]`	class=logging.handlers.RotatingFileHandler, level=DEBUG, formatter=standardFormatter, args=('py_module_service.log', 'a', 1048576, 10)
`[formatter_standardFormatt`	format=%(asctime)s.%(msecs)03d|%(levelname)s|%(message)s, datefmt=%Y-%m-%d %H:%M:%S

Note: `consoleHandler` - Responsible for logs that are been shown as console output, `handler_consoleHandler`: Handler that supplies additional configuration for console output logs. For example: * Class: Handler category, in this case it will be StreamHandler. * Level: Required log level (Supported levels: ERROR / WARNING / INFO / DEBUG). * Formatter: In which format the logs will be shown. * args: Arguments which are required for the handler class (keep as it is in the example).

`fileHandler` - Responsible for logs being exported to a log file, `handler_fileHandler`: Handler that supplies additional configuration for log files exportation: * Class: Handler category, in this case it will be RotatingFileHandler. * Level: Required log level (Supported levels: ERROR / WARNING / INFO / DEBUG). * Formatter: In which format the logs will be shown. * args: Arguments which are required for the handler class (keep as it is in the example). Based on the supplied example:

- `'py_module_service.log'` - The location + file name of generated log file.
- `'a'` - Means appending on the same file.
- `1048576` - Maximum file size: 1,048,576 bytes (1 MB).
- `10` - Retains 10 log files; older logs are deleted via rotation.

`formatter_standardFormatter`: Used for declaring formats for different uses: * `format=%(asctime)s.%(msecs)03d|%(levelname)s|%(message)s` - The log file that will be shown / exported. * `datefmt=%Y-%m-%d %H:%M:%S` - Used for internal usage. 'message' part contains statement details, including connection & statement ids, in order to have correlation also with Sqream logs.

Log format: `<datetime>|<log_type>|<connection_id>|<statement_id>|<log_message>`

Log output example: Python module logs also contain Connection & Session id information, in order to understand what was Sqream's executed statement that triggered those logs. Also, Python module's logs are on Python module execution level. In case of multiple execution on the same statement it is possible to correlate between both sides.

In SQream:

```
0 ✓ sagib@sagib-rh8 ~/branches/develop $ (sagib_add_pymodule_logs_20250415) cl
Interactive client mode
To quit, use ^D or \q.

master=> create or replace module pm options(path='/home/sagib/py_udf.py', entry_point
s=[[name='upper', param_types=[text], return_type=text, gpu_acceleration=true],[name='
lower', param_types=[text], return_type=text, gpu_acceleration=true]]);
executed
time: 0.554395s
master=> create or replace table t (x text);
executed
time: 0.609981s
master=> insert into t values ('aA'),('Bb'),('cCc') ;
executed
time: 0.574516s
master=> select pm.upper(x) as x1,  pm.lower(x) as x2  from t;
AA,aa
BB,bb
CCC,ccc
3 rows
```

Show node info: (Can see statement's python execution on node id level):

```
master=> select show_last_node_info();
0,1,PushToNetworkQueue  ,3,1,3,2025-11-17 22:06:42,-1,,,,2,0.0009
0,2,Rechunk             ,3,1,3,2025-11-17 22:06:42,1,,,,2,0.0000
0,3,GpuToCpu            ,3,1,3,2025-11-17 22:06:42,2,,,,2,0.0001
0,4,ReorderInput        ,3,1,3,2025-11-17 22:06:42,3,,,,2,0.0000
0,5,sqream::PythonModule,3,1,3,2025-11-17 22:06:42,4,,,,2,0.0243
0,6,sqream::PythonModule,3,1,3,2025-11-17 22:06:42,5,,,,2,0.0943
0,7,ReorderInput        ,3,1,3,2025-11-17 22:06:42,6,,,,2,0.0000
0,8,GpuDecompress       ,3,1,3,2025-11-17 22:06:42,7,,,,2,0.0000
0,9,CpuToGpu            ,3,1,3,2025-11-17 22:06:42,8,,,,2,0.0002
0,10,Rechunk             ,3,1,3,2025-11-17 22:06:42,9,,,,2,0.0000
0,11,CpuDecompress       ,3,1,3,2025-11-17 22:06:42,10,,,,2,0.0000
0,12,ReadTable           ,3,1,3,2025-11-17 22:06:42,11,0MB,,master.public.t,2,0.0010
```

Python module logs:

```
2025-11-17 21:39:37.873|INFO|c0|s7|New request: { module uid: conn0_stmt7_node6, module_path: /home/sagib/py_udf.py, func_name: upper, arg t
ypes: [bool_, int32, uint64, object_], return types: object_, rows: 3, offsets: 2048, 6656, 10240, 9216}
2025-11-17 21:39:37.896|INFO|c0|s7|New request: { module uid: conn0_stmt7_node5, module_path: /home/sagib/py_udf.py, func_name: lower, arg t
ypes: [bool_, int32, uint64, object_], return types: object_, rows: 3, offsets: 2048, 6656, 10240, 9216}
2025-11-17 21:39:37.915|INFO|c0|s7|Closing request
2025-11-17 21:39:37.943|INFO|c0|s7|Closing request
```

Note: Logs mentioned in this example can be shown both on console / exported to log file. `module_uid`: Python module execution unique identifier - contains: * Connection ID * Statement ID * Node ID (Execution tree identifier from show node info)

1.8 Python Functions

Python functions in SQream allow you to execute custom Python logic on your data and return the results. This functionality integrates the power of Python's data processing libraries (like Pandas) directly into your SQL queries, supporting two main types: **Table Functions** (which return a table) and **Scalar Functions** (which return a single value).

1.8.1 1. Syntax Overview

Python Table Functions (PTFs) and Python Scalar Functions (PSFs) are based on creating a Module.

1.8.1.1 Module Creation:

```
CREATE MODULE module_name OPTIONS (
        PATH = 'module.py',
        ENTRY_POINTS = [
                [
                        NAME = 'function_name',
                        ARGUMENTS [type1, type2, ...] | LIKE table_name,
                        RETURNS TABLE (col1 type1, col2 type2, ...) | LIKE table_name␣
↪| SCALAR type,
                        LITERAL_PARAMETERS = number,
                        GPU = true|false
                ]
        ]
);
```

Syntax	Description	Mandatory/Optional
PATH	Specifies the file path to your Python script on the server.	Mandatory
ENTRY_POI	A list of the Python functions within the script that can be called from SQream.	Mandatory
NAME	The name of the Python function.	Mandatory
ARGUMENTS	Can specify explicit types `[int, text, float]` or reference existing table structure `LIKE table_name`.	Mandatory
RETURNS	The RETURNS clause supports the following options: 1. `TABLE (column_name data_type, ...)` – Defines an explicit table structure. 2. `LIKE table_name` – Inherits structure from an existing table. 3. `SCALAR data_type` – Specifies a scalar return type.	Mandatory
GPU=true/	Whether the function executes on GPU or CPU. Default is CPU.	Optional
LITERAL_PA	Number of literal parameters the function accepts.	Optional

Notes:

1. Input Casting

The system applies automatic type casting to all input arguments.

2. Output Casting

- All return types are automatically marked as nullable (isNullable = true).
- Output columns are typed according to the function's declared return specification.

3. Literal Parameters

- Literal values can be passed to table functions.
- Literal parameters are stored as Seq[String] and provided to the function at execution time.

4. Dataframe Input Parameter to the Python Function

Both PTFs and PSFs **require** a dataFrame to be passed as the input parameter to the Python function.

1.8.1.2 Python Table Functions (PTFs): The <function_clause>

This clause defines the execution of your Python Table Function. It has the following structure:

```
table(<table_function>(cursor(<sub_query>), <literal_param>*));
```

Syntax	Description	Mandatory/Optional
table()	This is the main function wrapper that tells SQream to execute the Python Table Function and treat its result as a relation (a table).	Mandatory
table_functic	This is the fully qualified name of the Python function, including the module name. For example, arr_varif.final_array.	Mandatory
cursor(<sub_q	This is an optional argument that passes the result of a subquery (any valid SELECT statement) to your Python function. The data is provided to the Python function as a Pandas DataFrame.	Optional
<literal_param	These are optional string literals that are passed as additional arguments to your Python function. They must be defined in the module's literal_parameters option and will be cast to strings in the Python code.	Optional

1.8.1.3 Python Scalar Functions (PSFs): The <scalar_function>

A Python Scalar Function is called directly by its fully qualified name and accepts one or more arguments, which can be **column names** or **literal values**.

```
<module_name>.<scalar_function>([<arg1>, <arg2>, ...<literal_params>*]);
```

Syntax	Description	Mandatory/Optional
<module_name>	See the section for Python Module	Mandatory
<scalar_fu	The fully qualified name of the Python Scalar Function	Mandatory
[<arg1>, <arg2>, …]	The arguments passed to the function. These can be columns from the queried table (e.g., t.my_column) or literal values (e.g., 10, 'hello').	Optional
<literal_param	These are optional string literals that are passed as additional arguments to your Python function. They must be defined in the module's literal_parameters option and will be cast to strings in the Python code.	Optional

1.8.1.4 Example: Defining a Module with Both Function Types

```
CREATE OR REPLACE MODULE my_funcs
    OPTIONS(
path='/home/sqream/udf/my_functions.py',
entry_points =
[
    -- PTF Definition
    [
        name = 'process_data_table',
        arguments [text, int],
        returns table (processed_text text, processed_value int),
        gpu=true
    ],
    -- PSF Definition
    [
        name = 'add_one',
        arguments [int],
        returns int,
        gpu=false
    ]
]
    );
```

1.8.2 2. Usage Examples

1.8.2.1 Example 1: Python Scalar Function (PSF)

This example demonstrates calling a PSF in the SELECT list.

- **Python Function:**

```
def multiply(df):
"""
Takes a dataframe containing 'number' and 'multiplier' columns,
calculates the product row-by-row, and returns the modified dataframe.
"""
# Create a copy to ensure we don't modify the original data unintentionally
df_copy = df.copy()

# Vectorized multiplication: Multiplies aligned rows instantly
df_result = df_copy['number'] * df_copy['multiplier']

return df_result
```

- **Creating the Module in Sqream:**

```
CREATE OR REPLACE MODULE my_mod1
OPTIONS(
    PATH='/tmp/py_udf.py',
    ENTRY_POINTS=[
        [
            NAME = 'multiply',
            ARGUMENTS [int, int],
            RETURNS SCALAR int,
            GPU = false
```

(continues on next page)

(continued from previous page)

```
        ]
    ]
);
```

- **How to use the Module?**

```
create or replace table t (number int, multiplier int);
INSERT INTO t VALUES ( 1, 2 ),( 2, 2 );
select my_mod1.multiply(number, multiplier) as multiple_results from t;
```

- **Results:**

multiple_results
2
4

1.8.2.2 Example 2: Passing Literal Parameters For Python Scalar Function (PSF)

This example demonstrates how to pass literals to PSF.

- **Python Function:**

```
    def multiply(df,literals_map):
    """
Takes a dataframe containing 'number' and 'multiplier' columns,
calculates the product row-by-row, and returns the modified dataframe.
"""
num1= int(literals_map['0'])
num2= int(literals_map['1'])

df_copy = df.copy()

df_result = df_copy['number'] * df_copy['multiplier'] * num1 * num2
```

- **Creating the Module in Sqream:**

```
CREATE OR REPLACE MODULE my_mod2
OPTIONS (
                PATH = '/app/scalar_multipy_function_with_literals.py',
                ENTRY_POINTS = [
                                [
                                                NAME = 'multiply',
                                                ARGUMENTS [int, int],
                                                LITERAL_PARAMETERS = 2,
                                                RETURNS SCALAR int,
                                                GPU = false
                                ]
                ]
);
```

- **How to use the Module?**

```
create or replace table t (number int, multiplier int);
INSERT INTO t VALUES ( 2, 4 ),( 5 , 7 );
select my_mod2.multiply(number, multiplier,'10','20') as multiple_results from t;
```

- **Results:**

multiple_results
1600
7000

1.8.2.3 Example 3: Python Table Function (PTF)

This example demonstrates how to use cursor() to pass an entire table's data to a PTF.

- **Python Function:**

```
# Defined in 'my_functions.py'
def last_column(df):
df_new = df.iloc[:, -1:]
return df_new
```

- **Creating the Module in Sqream:**

```
CREATE OR REPLACE MODULE my_mod3
OPTIONS (
    PATH = '/app/my_functions1.py',
    ENTRY_POINTS = [
        [
            NAME = 'last_column',
            ARGUMENTS [date, datetime, text],
            RETURNS TABLE (col_name text),
            GPU = true
        ]
    ]
);
```

- **How to use the Module?**

```
CREATE OR REPLACE TABLE t (xdate DATE, xdatetime DATETIME, xtext TEXT);
INSERT INTO t VALUES (DATE '2025-09-11', DATETIME '2025-09-11 14:30:00', 'some sample␣
↪text');
SELECT * FROM table(my_mod3.last_column(CURSOR(SELECT * FROM t)));
```

- **Results:**

col_name
some sample text

Note: The `select * from t` subquery passes the t table to the function.

1.8.2.4 Example 4: Passing Literal Parameters For Python Table Function (PTF)

This example demonstrates how to pass literals to PTF.

- **Python Function:**

```
    # Defined in 'my_functions.py'
def add_literal_column(df, literals_map ):
df_new = df.copy()
df_new["col4"] = literals_map['0']
df_new["col5"] = literals_map['1']
return df_new
```

- **Creating the Module in Sqream:**

```
CREATE OR REPLACE MODULE my_mod4
OPTIONS (
    PATH = '/app/passing_literals.py',
    ENTRY_POINTS = [
        [
            NAME = 'add_literal_column',
            ARGUMENTS [boolean, int, date],
            LITERAL_PARAMETERS = 2,
            RETURNS TABLE (col1 boolean, col2 int, col3 date, col4 text, col5 int),
            GPU = true
        ]
    ]
);
```

- **How to use the Module?**

```
CREATE OR REPLACE TABLE t (col1 boolean, col2 int, col3 date);
INSERT INTO t VALUES (0, 1, '2025-09-11'),(1, 2, '2027-01-01');

SELECT col1, col2, col3, col4, (col5+5) FROM TABLE( my_mod4.add_literal_column( ␣
↪CURSOR(SELECT * FROM t), 'Text1', '1000') );
```

- **Results:**

col1	col2	col3	col4	EXPR$4
0	1	2025-09-11	Text1	1005
1	2	2027-01-01	Text1	1005

1.8.2.5 Example 5: Join Statment on a Python Table Function (PTF)

In the following example, the employees table stores employee attributes, while the sales_orders table contains sales records, including the employee responsible for each sale and the corresponding sale amount. A standard join between these two tables allows us to retrieve metrics such as the total number of sales per employee and the overall sales amount. To enhance this data, we may want to convert the sales amount from USD to EUR. The Parameterized Table Function (PTF) performs this enrichment by applying the appropriate conversion rate, which is supplied by the user at runtime. The PTF returns the enriched dataset with the total amount expressed in EUR.

- **Python Function:**

```
# Defined in 'my_functions.py'
def convertAmountBasedOnRate(df, literals_map):
    df_new = df.copy()
    rate = literals_map['0']

    if 'totalamount' in df_new.columns:
        df_new["ConvertedAmount"] = df_new['totalamount'] * rate
    else:
        # Handle case where the expected column isn't present
        print("Warning: 'totalamount' column not found. 'commission' column not added.
↪")

    return df_new
```

- **Creating the Module in Sqream:**

```
CREATE OR REPLACE MODULE my_mod5
OPTIONS (
    PATH = '/app/my_functions.py',
    ENTRY_POINTS = [
        [
            NAME = 'convertAmountBasedOnRate',
            ARGUMENTS [int, int, date, double],
            LITERAL_PARAMETERS = 1,
            RETURNS TABLE (orderid int, employeeid int, orderdate date, totalamount␣
↪double, ConvertedAmount double),
            GPU = true
        ]
    ]
);
```

- **How to use the Module?**

```
create table employees (EmployeeId int , FirstName text, LastName text, HireDate date,
↪ DepartmentId int);

insert into employees values (101,'Alex','Johnson','2023-01-15','3'),(102,'Sarah',
↪'Chen','2020-07-01','1'),
(103,'David','Lee','2024-11-20','4'),(104,'Emily','Smith','2021-10-25','4'),(105,'Ryan
↪','Garcia','2023-05-10','2');

create or replace table sales_orders (OrderId int, EmployeeId int, OrderDate date,␣
↪TotalAmount double);

insert into sales_orders values (5001,102,'2025-02-10',1250.00),
                                (5002,101,'2025-05-01',890.50),
                                (5003,102,'2025-09-15',3400.00),
                                (5004,104,'2025-08-22',520.25),
                                (5005,101,'2025-11-05',150.00),
                                (5006,102,'2025-10-01',1800.00),
                                (5007,104,'2025-06-18',985.00),
                                (5008,105,'2025-03-20',2500.00);

SELECT
    emp.EmployeeId AS "EmployeeId",
    COUNT(*) AS "NumberOfSales",
    SUM(convSales.totalamount) AS "TotalSalesAmountUsd",
```

(continues on next page)

(continued from previous page)

```
    SUM(convSales.ConvertedAmount) AS "TotalSalesAmountEur"
FROM
    employees AS emp
JOIN
    TABLE(
        my_mod5.convertAmountBasedOnRate(
            CURSOR(SELECT * FROM sales_orders),
            '0.86'
        )
    ) AS convSales
    ON emp.EmployeeId = convSales.EmployeeId
GROUP BY
    emp.EmployeeId
ORDER BY
    NumberOfSales DESC;
```

- **Results:**

EmployeeId	NumberOfSales	TotalSalesAmountUsd	TotalSalesAmountEur
102	3	6450.0	5547.0
101	2	1040.5	894.83
104	2	1505.25	1294.515
105	1	2500.0	2150.0

1.8.3 3. Return Values

- **Python Table Function(PTF):** Your Python function **must return a Pandas DataFrame**. The column names and data types of this DataFrame must exactly match the schema defined in the returns table(…) clause of the module's entry point.
- **Python Scalar Function(PSF):** Your Python function **must return a single, non-DataFrame Python value** (e.g., an integer, string, or float). This value will be automatically converted to the single SQL data type defined in the returns <data_type> clause of the module's entry point.

1.9 Catalog Tables

Catalog tables provide metadata about the machine learning models and Python modules created within AISQream, allowing users to query and understand the properties of these stored objects.

1.9.1 Catalog Table: ml_models

Contains all ML models with general common information:

–# DDL

```
SELECT get_ddl(sqream_catalog.ml_models);
```

```
create table "master"."sqream_catalog"."ml_models" (
   "model_id" bigint not null check ('CS ""flat""'),
   "database" text(32) not null check ('CS ""flat""'),
```

(continues on next page)

(continued from previous page)

```
   "schema" text(32) not null check ('CS ""flat""'),
   "name" text(32) not null check ('CS ""flat""'),
   "ml_alg" text(32) not null check ('CS ""flat""')
);
```

–# ml_models

```
SELECT * FROM sqream_catalog.ml_models;
```

```
10,master,public,m1,LINEAR_REG
11,master,public,ml_xgb_catalog_test,XGBOOST
8,master,s,m,LINEAR_REG
9,master,public,m,PY_MODEL
```

1.9.2 Catalog table: ml_linear_reg

Contains all Linear Regression models, with more detailed information specifically for Linear Regression:

–# DDL

```
SELECT get_ddl(sqream_catalog.ml_linear_reg);
```

```
create table "master"."sqream_catalog"."ml_linear_reg" (
   "model_id" bigint not null check ('CS ""flat""'),
   "database" text(32) not null check ('CS ""flat""'),
   "schema" text(32) not null check ('CS ""flat""'),
   "name" text(32) not null check ('CS ""flat""'),
   "ml_alg" text(32) not null check ('CS ""flat""'),
   "initial_algorithm" text(32) not null check ('CS ""flat""'),
   "gd_optimizer" text(32) not null check ('CS ""flat""'),
   "coefficients" text(32) not null check ('CS ""flat""'),
   "is_standardized" bool not null check ('CS ""flat""'),
   "learning_rate" real not null check ('CS ""flat""'),
   "epoch_count" int not null check ('CS ""flat""'),
   "loss_function" text(32) not null check ('CS ""flat""'),
   "tolerance" real not null check ('CS ""flat""'),
   "time" text(32) not null check ('CS ""flat""')
);
```

–# ml_linear_reg

```
SELECT * FROM sqream_catalog.ml_linear_reg;
```

```
10,master,public,m1,LINEAR_REG,SVD,ADAM,{0.500000|0.500000},1,0.0000,1,MSE,0.0000,
↪2025-02-06 15:10:55.150
8,master,s,m,LINEAR_REG,SVD,ADAM,{0.500000|0.500000},1,0.0000,1,MSE,0.0000,2025-02-06␣
↪15:10:42.121
```

1.9.3 Catalog table: ml_xgboost

Contains all XGboost models, with more detailed information specifically for XGboost:

–# DDL

```
SELECT get_ddl(sqream_catalog.ml_xgboost);
```

```
create table "master"."sqream_catalog"."ml_xgboost" (
   "model_id" bigint not null check ('CS ""flat""'),
   "database" text not null check ('CS ""flat""'),
   "schema" text not null check ('CS ""flat""'),
   "name" text not null check ('CS ""flat""'),
   "ml_alg" text not null check ('CS ""flat""'),
   "verbosity" text not null check ('CS ""flat""'),
   "booster" text not null check ('CS ""flat""'),
   "disable_default_eval_metric" text not null check ('CS ""flat""'),
   "tree_booster_parameters_eta" text not null check ('CS ""flat""'),
   "tree_booster_parameters_gamma" text not null check ('CS ""flat""'),
   "tree_booster_parameters_max_depth" text not null check ('CS ""flat""'),
   "tree_booster_parameters_min_child_weight" text not null check ('CS ""flat""'),
   "tree_booster_parameters_max_delta_step" text not null check ('CS ""flat""'),
   "tree_booster_parameters_subsample" text not null check ('CS ""flat""'),
   "tree_booster_parameters_sampling_method" text not null check ('CS ""flat""'),
   "tree_booster_parameters_colsample_bytree" text not null check ('CS ""flat""'),
   "tree_booster_parameters_colsample_bylevel" text not null check ('CS ""flat""'),
   "tree_booster_parameters_colsample_bynode" text not null check ('CS ""flat""'),
   "tree_booster_parameters_lambda" text not null check ('CS ""flat""'),
   "tree_booster_parameters_reg_lambda" text not null check ('CS ""flat""'),
   "tree_booster_parameters_alpha" text not null check ('CS ""flat""'),
   "tree_booster_parameters_reg_alpha" text not null check ('CS ""flat""'),
   "tree_booster_parameters_tree_method" text not null check ('CS ""flat""'),
   "tree_booster_parameters_scale_pos_weight" text not null check ('CS ""flat""'),
   "tree_booster_parameters_process_type" text not null check ('CS ""flat""'),
   "tree_booster_parameters_grow_policy" text not null check ('CS ""flat""'),
   "tree_booster_parameters_max_leaves" text not null check ('CS ""flat""'),
   "tree_booster_parameters_max_bin" text not null check ('CS ""flat""'),
   "tree_booster_parameters_num_parallel_tree" text not null check ('CS ""flat""'),
   "dart_booster_params_sample_type" text not null check ('CS ""flat""'),
   "dart_booster_params_normalize_type" text not null check ('CS ""flat""'),
   "dart_booster_params_rate_drop" text not null check ('CS ""flat""'),
   "dart_booster_params_one_drop" text not null check ('CS ""flat""'),
   "dart_booster_params_skip_drop" text not null check ('CS ""flat""'),
   "linear_booster_params_updater" text not null check ('CS ""flat""'),
   "linear_booster_params_feature_selector" text not null check ('CS ""flat""'),
   "linear_booster_params_top_k" text not null check ('CS ""flat""'),
   "objective" text not null check ('CS ""flat""'),
   "base_score" text not null check ('CS ""flat""'),
   "eval_metric" text not null check ('CS ""flat""'),
   "cut_off_top_positions" text not null check ('CS ""flat""'),
   "seed" text not null check ('CS ""flat""'),
   "seed_per_iteration" text not null check ('CS ""flat""'),
   "tweedie_variance_power" text not null check ('CS ""flat""'),
   "huber_slope" text not null check ('CS ""flat""'),
   "quantile_alpha" text not null check ('CS ""flat""'),
   "aft_loss_distribution" text not null check ('CS ""flat""'),
   "lambdarank_pair_method" text not null check ('CS ""flat""'),
   "lambdarank_num_pair_per_sample" text not null check ('CS ""flat""'),
```

(continues on next page)

(continued from previous page)

```
   "lambdarank_normalization" text not null check ('CS ""flat""'),
   "lambdarank_bias_norm" text not null check ('CS ""flat""'),
   "lambdarank_unbiased" text not null check ('CS ""flat""'),
   "ndcg_exp_gain" text not null check ('CS ""flat""'),
   "epoch_count" text not null check ('CS ""flat""')
);
```

–# ml_xgboost

```
SELECT * FROM sqream_catalog.ml_xgboost;
```

```
1,master,public,m2,XGBOOST,VERBOSITY_SILENT,BOOSTER_GBTREE,0,0.300000,0.000000,6,1,0,
↪1.000000,SAMPLING_METHOD_UNIFORM,1.000000,1.000000,1.000000,0.000000,0.000000,0.
↪000000,0.000000,TREE_METHOD_AUTO,1.000000,PROCESS_TYPE_DEFAULT,GROW_POLICY_
↪DEPTHWISE,0,0,1,SAMPLE_TYPE_UNSPECIFIED,NORMALIZE_TYPE_UNSPECIFIED,0.000000,0,0.
↪000000,UPDATER_UNSPECIFIED,FEATURE_SELECTOR_UNSPECIFIED,0,OBJECTIVE_BINARY_LOGISTIC,
↪0.000000,EVAL_METRIC_UNSPECIFIED,"""""",0,0,0.000000,0.000000,,AFT_LOSS_
↪DISTRIBUTION_UNSPECIFIED,LAMBDARANK_PAIR_METHOD_UNSPECIFIED,0,0,0.000000,0,0,100
```

1.9.4 Catalog table: py_modules

Contains all Python modules:

–# DDL

```
SELECT get_ddl(sqream_catalog.py_modules);
```

```
"module_id" bigint not null check ('CS ""flat""'),   "database_name" text(32) not␣
↪null check ('CS ""flat""'),   "module_name" text(32) not null check ('CS ""flat""'),
↪   "path" text(32) not null check ('CS ""flat""'),   "entry_points" text(32) not␣
↪null check ('CS ""flat""')
```

–# ml_linear_reg

```
SELECT * FROM sqream_catalog.py_modules;
```

```
22,master,pm,/home/sagib/py_udf.py,"{[name:func_name, args:[TEXT], ret_type:DATE,␣
↪gpu:true]}"
36,master,m1,/home/sagib/py_udf.py,"{[name:return_input2, args:[TEXT], ret_type:TEXT,␣
↪gpu:false]}"
```

1.9.5 Catalog table: py_module_permissions

Contains all Python modules permissions:

–# DDL

```
SELECT get_ddl(sqream_catalog.py_module_permissions);
```

```
database_name" text(32) not null check ('CS ""flat""'),   "module_id" bigint not null␣
↪check ('CS ""flat""'),   "role_id" bigint not null check ('CS ""flat""'),
↪"permission_type" int not null check ('CS ""flat""')
```

–# py_module_permissions

```
SELECT * FROM sqream_catalog.py_module_permissions;
```

```
master,0,8,11000
Master,0,8,11001
```

1.9.6 Catalog table: registered_algorithms

Contains all Python based AI models registered:

–# DDL

```
SELECT get_ddl(sqream_catalog.registered_algorithms);
```

```
"id" bigint not null check ('CS ""flat""'),
  "database_name" text(32) not null check ('CS ""flat""'),
  "alg_name" text(32) not null check ('CS ""flat""'),
  "alg_path" text(32) not null check ('CS ""flat""'),
  "train_method" text(32) not null check ('CS ""flat""'),
  "predict_method" text(32) not null check ('CS ""flat""')
```

–# registered_algorithms

```
SELECT * FROM sqream_catalog.registered_algorithms;
```

```
6,master,logistic_regression,/tmp/sagib.py,train,predict
```

1.10 Security Requirements

This section outlines the necessary user privileges for creating and managing Python modules within AISQream, ensuring controlled access and security for custom code execution.

- Python module creation is allowed to SUPERUSERs only.
- SUPERUSERs can grant EXECUTE permission to other non-SUPERUSERs: `GRANT EXECUTE ON MODULE module_name TO role;`

1.11 SQream Integration with Apache Iceberg

This document outlines SQream's integration with **Apache Iceberg**, a popular open-source table format designed for managing data lakes with transactional and schema evolution capabilities. The initial phases focus on establishing connectivity and enabling efficient read-only querying of existing Iceberg tables, followed by support for querying table metadata and time travel.

1.11.1 Overview of Apache Iceberg Integration

Apache Iceberg acts as a **table format** that manages the relationship between a logical table and its underlying data files (e.g., Parquet, ORC), along with metadata for versioning, statistics, and consistency. This makes it a crucial component in the **Data Lakehouse** concept.

1.11.1.1 Iceberg Architecture:

Iceberg uses a multi-layered metadata structure to track table state:

1. **Data Layer:** Contains the actual data in columnar file formats (e.g., **Parquet**) and **Delete Files** (for records that are logically deleted but physically still exist).
2. **Metadata Layer:** Tracks the table structure and its versions:

- **Metadata Files (JSON):** Stores the table's schema, partition schemes, and tracks the current and previous **Snapshots**.
- **Manifest Lists (AVRO):** Defines a Snapshot by listing all the **Manifest Files** that belong to that version.
- **Manifest Files (AVRO):** Track individual **Data Files** within a subset of the snapshot, including metadata for efficient data pruning (min/max values, null counts).

3. **The Catalog:** An external store (e.g., REST, AWS Glue) that maps a table name to its current **Metadata File** pointer, enabling transactional guarantees and multi-table semantics.

1.11.2 Connectivity to Iceberg

Connecting SQream to an external Iceberg REST Catalog.

1. **Create a Catalog Integration**

This step establishes the connection details to the Iceberg Catalog.

Syntax:

```
CREATE [ OR REPLACE ] CATALOG INTEGRATION <catalog_integration_name>
  OPTIONS (
        CATALOG_SOURCE = 'ICEBERG_REST',
        REST_CONFIG = (
          CATALOG_URI = '<rest_api_endpoint_url>',
          prefix = '<prefix to append to all API routes>',
          endpoint = '<file system endpoint uri>',
          access_key_id = "<access key>",
          secret_access_key = "<secret key>",
          region = "<region>"
        )
  );
```

Key Parameters:

Parameter	Description
CATALOG_SOURCE	Must be set to 'ICEBERG_REST' (default).
CATALOG_URI	The endpoint URL for the Iceberg REST Catalog API.

Usage Example:

```
CREATE OR REPLACE CATALOG INTEGRATION t_iceberg
  OPTIONS (
        CATALOG_SOURCE = 'ICEBERG_REST',
        REST_CONFIG = (
          CATALOG_URI = 'http://192.168.5.82:8181',
          prefix = 's3://warehouse/',
          endpoint = 'http://192.168.5.82:9000',
          access_key_id = 'admin',
          secret_access_key = 'password',
          region = 'us-east-1'
        ));
```

2. **Create a Foreign Database**

This links the new Catalog Integration to a database object within SQream.

Syntax:

```
CREATE FOREIGN DATABASE <database_name> catalog integration <catalog_integration_name>
↪;
```

Usage Example:

```
CREATE FOREIGN DATABASE t_iceberg_db catalog integration t_iceberg;
```

Note:

- This can only be performed on an empty database.
- SQream DB automatically converts Iceberg identifiers (database, namespace, and table names) to lowercase. Therefore, you must use lowercase names, or explicitly quote any identifier that contains uppercase letters or special characters.

Limitations:

- **File Format:** Only **Parquet** is supported.
- **Operations:** SELECT, INSERT, and DDL operations (excluding ALTER) are supported. DELETE and UPDATE are not currently supported and will be introduced in future phases.
- **Advanced Features:** schema evolution, and transactional commands **are not supported**.

1.11.3 DDL Operations on an Iceberg Table

An Iceberg table can be created in Sqream with DDL support for the data types listed below. ALTER operations are currently not supported.

Syntax:

```
CREATE [OR REPLACE] ICEBERG TABLE <FOREIGN_DATABASE>.<NAMESPACE>.table_name
(
    col_name1 col_type1 [NULL | NOT NULL],
    col_name2 col_type2 [NULL | NOT NULL],
    ...
)
[OPTIONS (
    [TBLPROPERTIES = (...)]
```

(continues on next page)

(continued from previous page)

```
    [COMMENT = 'table_comment']
)]
[AS select_statement];
```

TBLPROPERTIES - Supported values:

- 'write.update.mode' = 'copy-on-write' | 'merge-on-read' (Default: 'copy-on-write')
- 'write.delete.mode' = 'copy-on-write' | 'merge-on-read' (Default: 'copy-on-write')

```
TRUNCATE [TABLE] <FOREIGN_DATABASE>.<NAMESPACE>.table_name;

DROP [TABLE] [IF EXISTS] <FOREIGN_DATABASE>.<NAMESPACE>.table_name [PURGE];
```

Note:

- `PURGE` (Optional): Permanently deletes the table's underlying physical data files, immediately bypassing any trash or time-travel retention policies.

Usage Examples:

```
-- create table
CREATE OR REPLACE ICEBERG TABLE t_iceberg_db.test_namespace.t
(
    id BIGINT,
    event_time TIMESTAMP,
    data TEXT,
    category TEXT
)
OPTIONS (
    TBLPROPERTIES = [
        'write.update.mode' = 'copy-on-write',
        'write.delete.mode' = 'copy-on-write'
    ]
);

-- create as select
CREATE OR REPLACE ICEBERG TABLE t_iceberg_db.test_namespace.t1
AS SELECT * FROM x;

-- truncate
TRUNCATE TABLE t_iceberg_db.test_namespace.t;

-- drop soft delete
DROP TABLE IF EXISTS t_iceberg_db.test_namespace.t;

-- drop with purge
DROP TABLE IF EXISTS t_iceberg_db.test_namespace.t PURGE;
```

Note:

- Namespace creation is currently not supported in Sqream and must be performed externally.
- Partitions are not supported at this stage.

1.11.3.1 Data Type Mapping

SQream supports most standard Iceberg data types:

Iceberg Type	SQream Type	Notes
boolean	BOOL	
int, long	INT, BIGINT	
float, double	REAL, DOUBLE	
decimal(P,S)	NUMERIC(P,S)	Precision ≤ 38.
date, timestamp	DATE, DATETIME	
timestamp_ns	DATETIME2	Nanosecond precision.
string	TEXT	Stored as UTF-8.

1.11.4 Querying an Iceberg Table

An Iceberg table behaves like a regular SQream table for **SELECT** operations. SQream automatically uses the Iceberg metadata and statistics (like min/max filtering) to prune irrelevant data files, improving performance.

```
SELECT * FROM t_iceberg_db.namespace.my_iceberg_table WHERE column_a > 100;
```

1.11.4.1 Time Travel

Apache Iceberg's Time Travel capability allows users to query a table as it existed at a specific point in time or at a specific version. This is achieved through Iceberg's snapshot-based architecture, which captures the full state of the table following every write operation.

Time Travel Core Concepts:

- **Snapshots:** A snapshot represents the state of a table at a point in time. Every commit (insert, update, delete, or overwrite) generates a new snapshot.
- **Snapshot IDs:** Each snapshot is assigned a unique 64-bit integer ID, providing a definitive reference for auditing or rollbacks.
- **Immutability:** Once a snapshot is created, the underlying data files associated with it are immutable. This ensures that historical queries remain consistent even as the "live" table continues to evolve.

Syntax:

```
SELECT <select_list> FROM <database>.<namespace>.<iceberg_table>
       [[ TIMESTAMP | VERSION ] AS OF [ timestamp| unix_timestamp | snapshot-id ]];
```

Parameter	Description
`TIMESTAMP`	The `TIMESTAMP AS OF` clause allows for temporal lookups by resolving the most recent snapshot that was successfully committed at or before the specified point in time.
`VERSION`	The `VERSION AS OF` clause allows for deterministic query execution by pinning the query plan to a specific, unique Snapshot ID.

Usage Examples:

```
SELECT * FROM t_iceberg_db.namespace.my_iceberg_table TIMESTAMP AS OF '2023-04-
↪11T18:06:36.289' WHERE column_a > 100;

SELECT * FROM t_iceberg_db.namespace.my_iceberg_table VERSION AS OF 1231234;

SELECT * FROM t_iceberg_db.namespace.my_iceberg_table VERSION AS OF␣
↪2583872980615177898;
```

1.11.4.2 Extended Metadata Queries

Querying Snapshots (.snapshots)

Shows all valid snapshots for a table, including the operation that created them.

Syntax:

```
SELECT * FROM <database>.<namespace>.<iceberg_table>.snapshots;
```

Column	Data Type	Description
committed_at	DATETIME2	Timestamp of committed snapshot.
snapshot_id	BIGINT	The unique identifier for the snapshot.
parent_id	BIGINT	The ID of the previous snapshot.
operation	TEXT	Type of operation that created the snapshot (e.g., append).
manifest_list	TEXT	The full path to the manifest list file.

Querying History (.history)

Shows the changes and lineage of snapshots for a table.

Syntax:

```
SELECT * FROM <database>.<namespace>.<iceberg_table>.history;
```

Column	Data Type	Description
made_current_at	DATETIME2	Timestamp of when the snapshot became current.
snapshot_id	BIGINT	Unique identifier for the snapshot.
parent_id	BIGINT	The ID of the snapshot that preceded this one.
is_current_ancestor	BOOL	Indicates if this snapshot is an ancestor of the current table state.

Querying Manifests (.manifests)

The manifests table returns a list of all manifest files that make up the current table state.

Syntax:

```
SELECT * FROM <database>.<namespace>.<iceberg_table>.manifests;
```

Column	Data Type	Description
content	INT	Type of manifest: 0 (Data) or 1 (Deletes).
path	TEXT	The full URI/path to the specific manifest file stored in your storage layer
length	BIG-INT	The size of the manifest file in bytes.
added_snapshot_id	BIG-INT	The ID of the snapshot that first introduced this manifest file to the table.
added_data_files_c	INT	The number of new data files that were added within this specific manifest.
exist-ing_data_files_cour	INT	The number of data files that already existed and were carried over into this manifest.
deleted_data_files_	INT	The number of data files marked as deleted in this manifest.
added_delete_files_	INT	The number of new delete files (position or equality deletes) added to the table in the snapshot that created this manifest.
exist-ing_delete_files_co	INT	The number of previously existing delete files that are still active and were carried over into this manifest from earlier snapshots.
deleted_delete_files	INT	The number of delete files marked as removed in this manifest

Querying Files (.files)

The files table (often called the files metadata view) returns a granular list of every individual data file (e.g., Parquet, Avro, or ORC) currently tracked by the table.

Syntax:

```
SELECT * FROM <database>.<namespace>.<iceberg_table>.files;
```

Column	Data Type	Description
content	INT	Refers to type of content stored by the data file: 0 (Data), 1 (Position Deletes), 2 (Equality).
file_path	TEXT	Full file path and name
file_format	TEXT	Format, e.g. PARQUET.
spec_id	INT	Refers to the partition specification that a particular data file adheres to.
record_cou	BIGINT	Number of rows.
file_size_in	BIGINT	Size of file.
split_offset	AR-RAY[BIGI	A list of byte offsets within the file where it can be safely split for parallel reading. For example, in a large Parquet file, these offsets point to the start of row groups.
equal-ity_ids	AR-RAY[INT]	Used specifically for Equality Delete files. Field IDs of the columns used to determine if a row is deleted.
sort_order_	INT	The identifier for the specific Sort Order applied to the data within this file. Maps back to metadata to optimize join and aggregation performance.

Usefull Example:

In addition to querying Iceberg metadata tables, you can also join them with each other.

```
SELECT *
FROM t_iceberg_db.namespace.my_iceberg_table1.manifests a
JOIN t_iceberg_db.namespace.my_iceberg_table2.snapshots b
    ON a.added_snapshot_id = b.snapshot_id;
```

1.11.5 Write operations on an Iceberg Table

Iceberg tables support data ingestion through three primary mechanisms: manual row insertion via INSERT, bulk loading from existing datasets using INSERT INTO … SELECT, and atomic table creation with data population via CREATE TABLE AS SELECT (CTAS).

Syntax:

```
-- Standard Append (Values)
INSERT [INTO] <FOREIGN_DATABASE>.<NAMESPACE>.table_name
        [(col_name [, ...])]
        VALUES (expression [, ...]) [, (expression [, ...]), ...];

-- Insert from Select Statement
INSERT [INTO] <FOREIGN_DATABASE>.<NAMESPACE>.table_name
        <select_statement>;
```

Usage Examples:

```
CREATE or replace ICEBERG TABLE test_foreign_db.test_namespace.all_types (
        b bool,
        i int,
        bi bigint,
        d double,
        n numeric(20, 10),
        ts timestamp,
        dt date,
        dtm datetime,
        dt2 datetime2,
        txt text
);

--standard insert
insert into test_foreign_db.test_namespace.all_types values (
        1,
        123,
        5632323,
        2.5,
        1234567890.1234567890,
        '2019-12-07 23:04:26' ,
        '1999-11-05',
        '1955-11-05 01:24:00.000',
        '1999-11-05 01:24:00.000666333',
        'test_data');

--bulk insert
insert into test_foreign_db.test_namespace.all_types select * from test_foreign_db.
↪test_namespace.all_types;

--create iceberg table from existing internal table
CREATE ICEBERG TABLE test_foreign_db.test_namespace.all_types_dest as select * from␣
↪all_types_internal_table;
```

1.12 Access Control

1.12.1 Permissions

SQream's Python Module allows users to integrate custom Python code and functions directly. This section describes the permissions required for the AI/ML features. For all other permissions - you can find the full permissions details here SQream documentation.

1.12.1.1 Objects

- Module - A Python module that contains custom Python code and functions that can be invoked from Sqream.
- Algorithm - A registered machine-learning recipe that defines how to train a model and run inference. Sqream provides native support for Linear Regression and XGBoost, and additional algorithms can be provided via register algorithm.
- Model - A trained machine learning artifact created from an algorithm and data. models can be used for inference.

Permission	Description
Database	
`CREATE MODULE`	Ability to create a new module. The role that creates the module receives all permissions (EXECUTE, DDL)
`CREATE ALGORITHM`	Ability to register a new algorithm. The creator receives `USAGE` and `DDL` permissions
Schema	
`CREATE MODEL`	Ability to create a new model in the schema
Module	
`EXECUTE`	Ability to execute any function within a specified module
`DDL`	Ability to drop a module
`ALL`	Encapsulates both `EXECUTE` and `DDL` permissions
Algorithm	
`USAGE`	Ability to create a new Python model based on the algorithm
`DDL`	Ability to unregister an algorithm
`ALL`	Encapsulates both `USAGE` and `DDL` permissions
Model	
`EXECUTE`	Ability to use a specified model for inference with the `model_predict` function
`DDL`	Ability to remove a model from the database
`ALL`	Ability to drop a model

1.12.1.2 Syntax

Permissions may be granted or revoked using the following syntax.

1.12.1.2.1 GRANT

```
-- Grant create module permissions to a role:
GRANT {
CREATE MODULE
ON DATABASE <database> [, ...]
TO <role> [, ...]

-- Grant execute/ddl/all on module:
GRANT {
EXECUTE
| DDL
| ALL
ON MODULE <module> [, ...]
TO <role> [, ...]

-- Grant create algorithm permissions to a role:
GRANT {
CREATE ALGORITHM
ON DATABASE <database> [, ...]
TO <role> [, ...]

-- Grant usage/ddl/all on algorithm:
GRANT {
USAGE
| DDL
| ALL
ON ALGORITHM <algorithm> [, ...]
TO <role> [, ...]

-- Grant create model at the schema level:
GRANT {
CREATE MODEL
ON SCHEMA <schema> [, ...]
TO <role> [, ...]

-- Grant execute/ddl/all on model:
GRANT {
EXECUTE
| DDL
| ALL
ON MODEL <schema>.<model> [, ...]
TO <role> [, ...]
```

1.12.1.2.2 REVOKE

```
-- Revoke create module permissions from a role:
REVOKE {
CREATE MODULE
ON DATABASE <database> [, ...]
FROM <role> [, ...]

-- Revoke execute/ddl/all permissions on module:
REVOKE {
EXECUTE
```

(continues on next page)

(continued from previous page)

```
| DDL
| ALL
ON MODULE <module> [, ...]
FROM <role> [, ...]


-- Revoke create algorithm permissions to a role:
REVOKE {
CREATE ALGORITHM
ON DATABASE <database> [, ...]
FROM <role> [, ...]

-- Revoke usage/ddl/all on algorithm:
REVOKE {
USAGE
| DDL
| ALL
ON ALGORITHM <algorithm> [, ...]
FROM <role> [, ...]

-- Revoke create model at the schema level:
REVOKE {
CREATE MODEL
ON SCHEMA <schema> [, ...]
FROM <role> [, ...]

-- Revoke execute/ddl/all on model:
REVOKE {
EXECUTE
| DDL
| ALL
ON MODEL <schema>.<model> [, ...]
FROM <role> [, ...]
```

1.12.1.3 Examples

1.12.1.3.1 GRANT

Grant create module on db database to role_name:

```
GRANT CREATE MODULE ON DATABASE db TO role_name;
```

Grant execute/ddl/all on md module to role_name:

```
GRANT EXECUTE,DDL,ALL ON MODULE md TO role_name;
```

Grant create algorithm on db database to role_name:

```
GRANT CREATE ALGORITHM ON DATABASE db TO role_name;
```

Grant usage/ddl/all on algo ALGORITHM to role_name:

```
GRANT USAGE,DDL,ALL ON ALGORITHM algo TO role_name;
```

Grant create model on s1 schema to role_name:

```
GRANT CREATE MODEL ON SCHEMA s1 TO role_name;
```

Grant execute/ddl/all model mod1 to role_name:

```
GRANT EXECUTE,DDL,ALL ON MODEL s1.mod1 TO role_name;
```

1.12.1.3.2 REVOKE

Revoke create module on db database from role_name:

```
REVOKE CREATE MODULE ON DATABASE db FROM role_name;
```

Revoke execute/ddl/all on md module from role_name

```
REVOKE EXECUTE,DDL,ALL ON MODULE md FROM role_name;
```

Revoke create algorithm on db database from role_name:

```
REVOKE CREATE ALGORITHM ON DATABASE db FROM role_name;
```

Revoke usage/ddl/all on algo ALGORITHM from role_name:

```
REVOKE USAGE,DDL,ALL ON ALGORITHM algo FROM role_name;
```

Revoke create model on s1 schema from role_name:

```
REVOKE CREATE MODEL ON SCHEMA s1 FROM role_name;
```

Revoke execute/ddl/all model mod1 from role_name:

```
REVOKE EXECUTE,DDL,ALL ON MODEL s1.mod1 FROM role_name;
```