Concepts

SQream Technologies

Version 2019.2.1

Table of Contents

Concepts

Download as PDF

This section describes SQream's database concepts.

SQream DB Daemon

In SQream DB, the `sqreamd` or SQream daemon is the server that deals with most of the operations against a certain GPU. Certain installations may have several `sqreamd` running, each with their own GPU ID.

A `sqreamd` may run with a specific storage cluster, or be teamed up with others, sharing a shared cluster.

Storage Cluster

A SQream DB Storage Cluster is a collection of all stored objects:

- Databases
- Schemas
- Tables
- Columns
- External Tables
- User Defined Functions
- Roles

A server instance can only run against a single storage cluster at one time. Clusters *can be changed*, but require a restart of the daemon.

> **TIP:**
>
> A cluster will be created on installation, and shouldn't require any intervention during normal operation

Databases

A storage cluster may have many databases residing in it.

When you create an applicative connection (from a client or JDBC/ODBC) – you connect to a single database.

A database can have many Schemas and Tables.

> **TIP:**
>
> Create different databases for different use-cases

- To view existing databases, query the Catalog (information schema).
- To create a new database, query the CREATE DATABASE command.

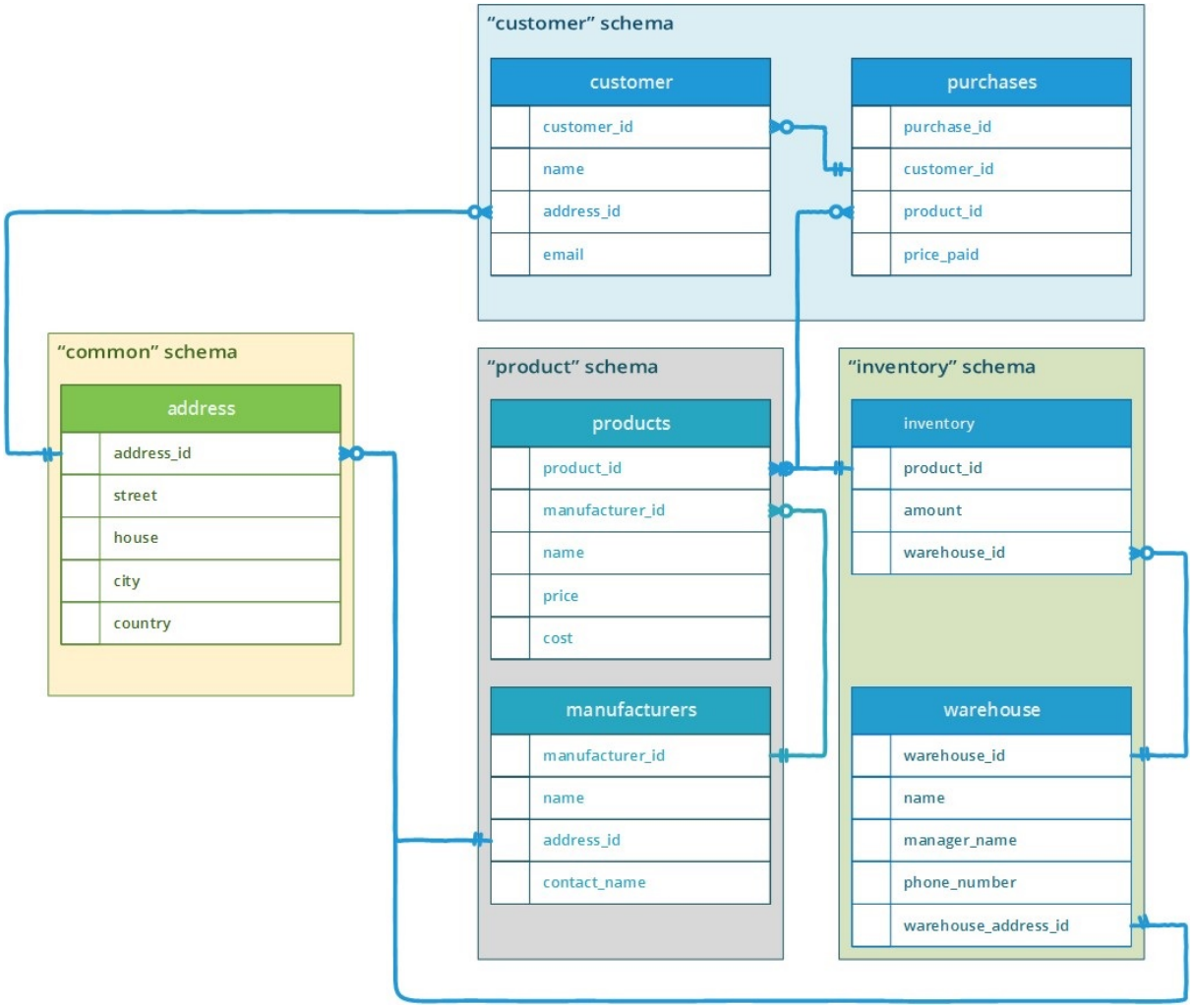
Schemas

Schemas are part of the effective table name. The default schema is called `public`.

- To view existing schemas, query the Catalog (information schema).
- To create a new schema, query the CREATE SCHEMA command.
- To manage schemas permissions, query Database Roles and Permissions.

TIP:

Use schemas to split up the data tables logically

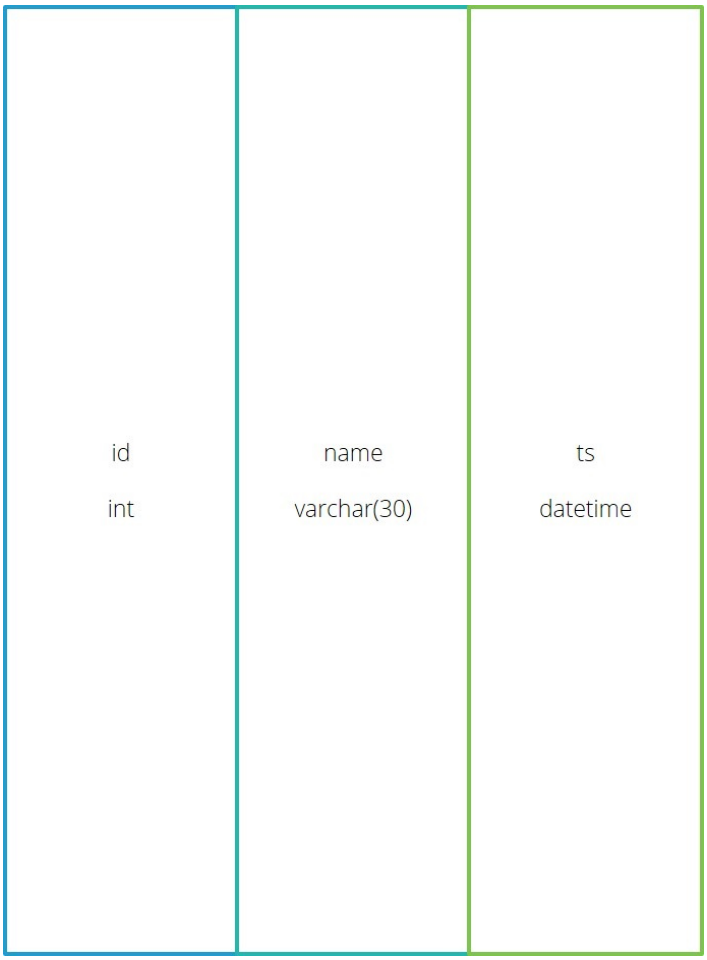


Example of schema usage

Tables

In SQream DB, a table is a set of data elements organized into horizontal rows and vertical columns. A table has a specified number of columns but can have any number of rows.

- To view tables, query the Catalog (information schema).
- To create a new table, query the CREATE TABLE command.
- To manage tables permissions, query Database Roles and Permissions.



Logical table 'customers'

In a row-oriented database, would be stored as:

```
1,John,1997-01-01 15:30:30;2,Jane,1998-05-06 00:00:00;...
```

In a columnar database, would be stored as:

```
1,2,3,4,5;
John,Jane,Sam,David,Susan;
1997-01-01 15:30:30,1998-05-06 00:00:00,1956-06-02 14:31:00,2006-
12-25 14:40:00,1975-10-21 02:20:00
```

TIP:

Instead of pulling the entire row (all columns) every time, only select the columns you need.
SQream DB automatically removes columns not necessary for the calculations.

Example

```
SELECT name, MAX(ts) FROM customers GROUP BY name;
```

Columns

A column is the element that makes up a table.

Each column has a specific type and name, that do not change. By physically organizing data by columns, SQream DB is able to scan and aggregate data for individual columns faster, because less data needs to be read. This makes columns highly suitable for analytical workloads.

External Tables

External tables are structured DDL that allow SQream to access data that which is stored outside the database in a none-SQream format, and to query it via SQL commands. Upon creation, the user should specify the external files format and location, and the needed table DDL. Once created, SQream will query the tables, as if they were regular tables.

- To view existing external tables, query the Catalog (information schema).
- To create a new external table, query the CREATE EXTERNAL TABLE.
- To manage external tables permissions, query Database Roles and Permissions.

User Defined Functions

User defined functions (UDF) are used by the DBA/BI/Data scientist to run custom made programs, written in Python, within SQream DB space. By using the UDF, the customers can extent the existing built-in operations in SQream, either as a row-by-row function, or as a utility function.

To read more about UDF, please see User Defined Functions in the SQL Reference Guide.

Chunks, Compression and Metadata

Chunks

SQream DB splits columnar data into chunks. The chunk size will determine the minimum bulk load into the GPU. For better performance, SQream DB rearranges previously loaded data into new chunks based on the configured **chunk size**. When loading data into SQream DB, each bulk load (either `INSERT INTO` or `COPY`), will generate its own chunks (with sizes up to the chunk size). Chunk size is a parameter at the cluster level. It must be set before the first insert to the cluster. It can also be set at the database level, before any tables are created. Default chunk size is 1 million rows. Ask your database administrator about setting the chunk size.

> **NOTE:**
>
> - The chunk size influences load/query time. Before tuning the parameter, consult your SQream account manager.

Compression specs

When `DEFAULT` compression spec (or no compression spec) is specified, each chunk may be compressed in a variety of different formats, based on the system's understanding. You may override the compression spec, but this is not recommended.

See Compression types in the SQL Manual for more information.

Metadata

SQream DB gathers and saves metadata information regarding the columns data at the chunk level during `COPY`. This information will serve the SQream optimizer while doing Data Skipping and other optimizations. Gathering metadata is automatic and transparent and requires no user intervention.

Catalog (information schema)

The SQream DB catalog or information schema consists of views that contain information about all database objects. This provides access to database metadata, column types, tables and their row-counts, etc.

The catalog structure is specific to SQream DB.

Querying the SQream catalog

The catalog is available from any database, under the schema called `sqream_catalog`. You may query the schema as you would any other table in the system.

> **NOTE:**
> You can not perform any other operations on the catalog, like `INSERT`, `DELETE`, ...

Examples:

```
demo_db=> SELECT * from sqream_catalog.tables;
```

Example result for a demo database

database_name	table_id	schema_name	table_name	row_count_valid	row_count	rechunker_ignore
demo_db	0	public	nation	1	25	0
demo_db	1	public	region	1	5	0
demo_db	2	public	part	1	20000000	0
demo_db	3	public	supplier	1	1000000	0
demo_db	4	public	partsupp	1	80000000	0
demo_db	5	public	customer	1	15000000	0
demo_db	6	public	orders	1	300000000	0
demo_db	7	public	lineitem	1	600037902	0

Examples for identifying delete predicates on tables

```
demo_db=> select t.table_name,d.* from sqream_catalog.delete_
predicates d
.> inner join sqream_catalog.tables t on
.> d.table_id=t.table_id;
```

Available catalog views

There are 3 categories of catalog views:

- Database object catalog
- Fine-grain storage catalog
- Role and permission catalog

Database object catalog

View name	Description
`sqream_catalog.databases`	All database objects in the current storage cluster
`sqream_catalog.schemas`	Schema objects in the database
`sqream_catalog.tables`	Table objects in the database
`sqream_catalog.columns`	Column objects in the current database
`sqream_catalog.views`	View objects in the database
`sqream_catalog.external_tables`	External table objects in the database
`sqream_catalog.udf`	User defined functions in the current database
`sqream_catalog.catalog_tables`	All catalog views available

Fine-grain storage catalog

View name	Description
`sqream_catalog.extents`	Extent storage objects in the database
`sqream_catalog.chunks`	Chunk storage objects in the database
`sqream_catalog.delete_predicates`	Logical delete predicates added to the compiler with a `DELETE` command

Role and permission catalog

View name	Description
`sqream_catalog.roles`	Roles (users) in the current databases
`sqream_catalog.role_memberships`	Roles membership
`sqream_catalog.table_permissions`	Tables and their assigned roles
`sqream_catalog.database_permissions`	Databases and their assigned roles
`sqream_catalog.schema_permissions`	Schemas and their assigned roles
`sqream_catalog.permission_types`	Permission types

Database object catalog

Databases view: sqream_catalog.databases

Column name	Type	Description
database_id	varchar	Database ID
database_name	varchar	Database name
default_disk_chunk_size	bigint	Storage chunk size (in number of rows)
default_process_chunk_size	bigint	Process chunk size (in number of rows)
rechunk_size	bigint	Internal use
storage_subchunk_size	bigint	Internal use
compression_chunk_size_threshold	bigint	Internal use

Schemas view: sqream_catalog.schemas

Column name	Type	Description
schema_id	varchar	Schema ID
schema_name	varchar	Schema name
schema_owner	varchar	Role who owns this schema
rechunker_ignore	bigint	Internal use

Tables view: sqream_catalog.tables

Column name	Type	Description
database_name	varchar	Owning database name
table_id	varchar	Table ID
schema_name	varchar	Owning schema name
table_name	varchar	Table name
row_count_valid	bool	See warning below
row_count	bigint	Number of rows in the table
rechunker_ignore	int	Internal use

WARNING:

When `row_count_valid` is 0 (after a DELETE operation), the row count may be inaccurate. To get the accurate row-count, run

```
SELECT COUNT(column) FROM table;
```

Columns view: sqream_catalog.columns

Column name	Type	Description
database_name	varchar	Owning database name
schema_name	varchar	Owning schema name
table_id	varchar	Owning table ID
table_name	varchar	Owning table name
column_id	int	Column ID
column_name	varchar	The column name
type_name	varchar	Column type
column_size	bigint	Column data size in bytes
has_default	int	Indicates whether or not the column has a default
default_value	varchar	Indicates the default column value
compression_strategy	varchar	User-overridden compression strategy
created	varchar	Creation date
altered	varchar	Last alter date

Views view: sqream_catalog.views

Column name	Type	Description
view_id	varchar	View ID
view_schema	varchar	Owning schema name
view_name	varchar	The view name
view_data	varchar	Internal use
view_query_text	varchar	Full statement test that created this view

External tables view: sqream_catalog.external_tables

Column name	Type	Description
database_name	varchar	Owning database name
table_id	varchar	External table ID
schema_name	varchar	Owning schema name
table_name	varchar	External table name
format	int	0=CSV, 1=Parquet
created	varchar	Creation data as a string

User defined functions: sqream_catalog.udf

Column name	Type	Description
database_name	varchar	Owning database name
function_id	varchar	The function ID
function_name	varchar	The function name

Fine-grain storage catalog

Extent view: sqream_catalog.extents

Column name	Type	Description
database_name	varchar	Owning database name
table_id	varchar	Owning table ID
column_id	bigint	Owning column ID
extent_id	bigint	The extent ID
size	bigint	Size of the extent in MB
path	varchar	Full path to the extent file on disk

Chunks view: sqream_catalog.chunks

Column name	Type	Description
database_name	varchar	Owning database name
table_id	varchar	Owning table ID
chunk_id	bigint	The chunk ID
rows_num	bigint	The amount of rows in this specific chunk
deletion_status	bigint	This chunk's deletion mark. 0 means keep, 1 means chunk needs partial deletion, 2 means delete entire chunk.

Delete predicates view: sqream_catalog.delete_predicates

Column name	Type	Description
database_name	varchar	Owning database name
table_id	varchar	Owning table ID
max_chunk_id	bigint	The highest chunk_id seen during the DELETE time
delete_predicate	varchar	The predicates added by the compiler (one predicate-

Column name	Type	Description
		statement per row in this view)

Role and permission catalog

Role view: sqream_catalog.roles

Column name	Type	Description
role_id	bigint	The role ID
name	varchar	Role name
superuser	bool	1 for superuser, 0 otherwise
login	bool	1 if the role has login permission, 0 otherwise
has_password	bool	Does this role have a password?
can_create_function	bool	Does this role have the permissions to create/revoke user defined functions?

Role membership view: sqream_catalog.role_memberships

Column name	Type	Description
role_id	int	The role ID
member_role_id	int	This role is member of role_id
inherit	bool	1 for inherit permission, 0 otherwise.

Table permission view: sqream_catalog.table_permissions

Column name	Type	Description
database name	varchar	Owning database name
table_id	bigint	Owning table ID
role_id	bigint	The role ID
permission_type	int	The permission type

Database permission view: sqream_catalog.database_permissions

Column name	Type	Description
database name	varchar	Owning database name
role_id	bigint	The role ID
permission_type	int	The permission type

Schema permission view: sqream_catalog.schema_permissions

Column name	Type	Description
database_name	varchar	Owning database name
schema_id	bigint	Owning schema ID
role_id	bigint	The role ID
permission_type	int	The permission type

Permission type view:sqream_catalog.permission_types

Column name	Type	Description
permission_type_id	bigint	The permission type ID
name	varchar	Permission name

Locks

SQream DB operates in two modes: **exclusive**, which sends a single operation at a time, and **inclusive** which is a multi operations mode. DDL operations are always exclusive.

DML are separated to **DELETE/TRUNCATE** as exclusive; and **INSERT** as inclusive. This allows multiple inserts into the table, but prevents multiple **DELETE** operations.

Querying (**SELECT** operations) can coexists with both DDL and DML.

Locking

SQream locks

Operation	Select	DML (Insert)	DML (Delete/Truncate)	DDL
Select	No lock	No lock	No lock	No lock
DML (insert)	No lock	No lock	No lock	Lock
DML (delete/truncate)	No lock	No lock	Lock	Lock
DDL	No lock	Lock	Lock	Lock

By default, when a session is requesting a lock on an object and the object is busy, SQream will wait 3 seconds before it return an error message. This wait time is defined in the configuration JSON. See the `statementLockTimeout` parameter in SQream Administrator Guide for more information.

> **NOTE:**
> DDL on an object will prevent other DDL/DML to wait on a lock on the same object.

For specific DDL operations, SQream uses global permissions that requires very short exclusive locks at the cluster level. Global permission will be used on operation such as **CREATE DATABASE/TABLE/ROLE**, **ALTER ROLE/TABLE**, **DROP ROLE/TABLE/DATABASE**, **GRANT/REVOKE**.

Monitoring locks

To view all existing locks in the SQream database use the utility function show_locks() :

Example

```
SELECT  show_locks();
```

Workload Manager

SQream will distribute work throughout the hardware resources to maximize the hardware utilization. By default, this distribution will be done in an equal manner. The DBA can change this setting and optimize the utilization to their needs by using SQream workload manager and defining each SQream instance to specific service/s. The specific service to connect to is defined in the session connection string, with the property 'service'. Default service name is: sqream (for more details, see each driver connection string specification).

Each SQream instance can serve multiple services, and each service can work with multiple SQream instances.

For an easy start, see Quick guide to using SQream Dynamic Workload Manager (DWLM).

Managing Services

Monitor services subscription

```
select show_subscribed_instances();
```

> **NOTE:**
> Instance ID is a unique identifier, defined by SQream at the installation, for each instance in SQream cluster.

Example

```
select    show_subscribed_instances('etl_service');
select    show_subscribed_instances();
```

Add services to an existing instance

```
select subscribe_service('instance_id', 'service_name'); ;
```

Example

```
select  subscribe_service('node_11',  'etl_service');
```

Remove services from an existing instance

```
select unsubscribe_service('instance_id', 'service_name'); ;
```

Example

```
select  unsubscribe_service('node_11',  'etl_service');
```

> **NOTE:**
> You cannot unsubscribe the last instance from an existing service that has working/waiting statements in its queue.

Utility Functions

Use the SQream Utility Functions (UF) to monitor and manage SQream cluster.

Utility functions cannot be used as a part of a query. Use the following syntax:

```
select utility_function_name();
```

To see the list of all existing UF:

```
select list_utility_functions();
```

To see the UF columns:

```
select show_uf_column_names('utility_function_name');
```

Example:

```
select show_uf_column_names('show_server_status');
```

Commonly Used UF:

Utility Function	Usage	Comment
`show_server_status`	Show all connections to the server and their current status	See quick guide to managing and monitoring SQream cluster
`show_node_info`	Show progress of a specific statement ID	
`show_cluster_nodes`	Show all the instances in SQream cluster	
`show_locks`	Show all current locks	
`show_conf`	Show current cluster internal parameter settings	
`get_chunk_size`	Show the current database chunk size	
`backup_storage`	Backup the cluster (and all its databases)	
`get_ddl`	Generate the table DDL command	See get metadata in SQL Reference

Utility Function	Usage	Comment
`get_view_ddl`	Generate the view DDL command	
`dump_database_ddl`	Generate DDL command for all tables and views in the DB	
`subscribe_service`	Subscribe a service to an instance	See Workload Manager & DWLM Quick Guide
`unsubscribe_service`	Unsubscribe a service to an instance	
`show_subscribed_instances`	List all instances/services in the cluster	
`list_saved_queries`	List all saved queries (ID & name)	See Saved Queries
`execute_saved_query`	Execute a specific saved query	
`drop_saved_query`	Drop a specific saved query	
`cleanup_chunks`	Physical delete of deleted rows at chunk level	See DELETE command
`cleanup_extents`	Physical delete of deleted rows at extent level	
`discard_results`	Option to run the select (or UF??) without showing the output	Example: `SELECT discard_results ('write your query' here);`

> **NOTE:**
> For utility functions that require the use of single quotation marks to wrap an SQL statement, you might find it useful to wrap the statement in '$$' to avoid quotation mark issues

Example

```
select discard_results($$ select c_custkey from customer where
c_name='John' $$);
```