Quick guide to performance tuning best practices

SQream Technologies

Version 2019.2.1

Table of Contents

Quick guide to performance-tuning best practices

The following guidelines will help you to optimize database use, reduce database size, and maximize performance.

These principles should be applicable in almost all cases. However, we strongly recommend verifying with your SQream DBA before complete implementation.

Loading Data

- SQream is intended for big data.

  For better performance, load in large bulks. Best loads will be in multiples of the chunk size (default = 1 million rows).

- Take advantage of data skipping.

  Due to the metadata that SQream keeps on the existing data, SQream can often do data skipping and eliminate wasted IO/GPU time.

  For better data skipping, sort the loaded data on the main columns you intend to filter in your queries, either before the load (in case of copy) or during (in case of insert/create as 'select from external table').

  Sorting the data will also be relevant for delete operations; if the data is fully/mostly sorted by the delete key column, the physical cleanup will be faster and use fewer resources.

> **NOTE:**
>
> SQream is optimized in such a way that sort during insert (command `INSERT INTO|CREATE table <AS> SELECT … FROM .. ORDER BY <column_list>`) will do a partial and faster sort based on the O/S RAM without using extra I/O.

Data Definition

- For short ASCII strings (up to 15 bytes), use VARCHAR datatype.

  > **NOTE:**
  > Non-ASCII characters must use NVARCHAR (as opposed to VARCHAR).

- For large strings columns (greater than 16 bytes) use NVARCHAR datatypes regardless of the character language.
- To avoid the need for explicit cast, use INT/BIGINT when possible (as opposed to TINYINT or SMALLINT).

  > **NOTE:**
  > When an integer reaches its datatype size limitation (tinyint 256, smallint 32767 etc.), SQream will return an overflow.

- Not Null constraint:

  Since SQream is a columnar database, it is recommended to define any column that logically cannot contain nulls as 'not null'.

  Example

```
Create table my_table (
       A int not null,
       B nvarchar(100),
       C datetime not null) ;
```

- Since SQream is a columnar database that can do high-speed JOINS, the issue of normalizing/not normalizing tables is not critical. Nevertheless, use the following guidelines for making that decision:
  - Normalization to fact/dimension (ratio of 1:N) will help to reduce row length size at the fact table, decrease database size, and optimize I/O and GPU utilization at query run time.

    It is highly recommended to use integers (int/bigint) as the join key and not string.

Fact Table:

ID	P_Date	Store_ID	Amount
1	<date>	1	<amount>
2	<date>	1	<amount>
3	<date>	2	<amount>
4	<date>	2	<amount>

Dimension table:

Store_ID	Store_Name
1	1_store_name
2	2_store_name
3	3_store_name
4	4_store_name

5	<date>	4	<amount>
6	<date>	1	<amount>
7	<date>	4	<amount>
8	<date>	1	<amount>
9	<date>	2	<amount>
10	<date>	4	<amount>

- Choose not to normalize when de-normalization can save join time without resulting in excessive storage requirements (when you have the advantage of high compression ratio and data duplication is not excessive).

Query Processing and SQL Usage

- Joins: To take advantage of GPU utilization, consider writing the queries in a way that allows reducing the fact tables before the join operation.

  Example

```
Example query:

select  store_name, sum(amount)
from dimention dim join fact on dim.store_id=fact.store_id
where p_date between '2018-07-01' and '2018-07-31'
group by  store_name;
Recommended query syntax to reduce fact table:
select store_name, sum_amount
from dimention as dim inner join
       (select sum(amount) as sum_amount, store_id
       from fact
       where p_date between '2018-07-01' and '2018-07-31'
       group by store_id
              ) as fact
on dim.store_id=fact.store_id;
```

- Use high selectivity hint when filtering out most of the table on a non-sorted column. See WHERE with HIGH_SELECTIVITY
- Due to the metadata that SQream keeps on the existing data, SQream can often do data skipping and and eliminate wasted IO/GPU time.

  For better data skipping, increase the use of sargable conditions to do as much filtering as possible in the WHERE clause: (=,>, <, >=, <=, BETWEEN, LIKE, IS [NOT] NULL, EXISTS, IN).

- When using INT or smaller datatype, CAST large results (such as SUM) to BIGINT to avoid overflows:

```
Select sum(cast(my_int_column as bigint)) from table;
```

- Whenever possible, use '`select count(*)`' rather than '`select count (column_name)`'

  `count(*)` allows SQream to read the metadata and to avoid unecessary calculations.

  Please note that in SQream the metadata is always up-to-date and there is no need to gather statistics.

  Example

```
> select count(*) from big
```

```
85598208
1 row
time: 0.065940s
```

- Query only the columns you need (unlike `select * from`).
  Since SQream is a columnar database, it will load all the columns in the statement.

  Example

```
Assuming the table has 20 columns and you need only 4:

Select col1, col2, col10 from my_table
where col7='my_predicate';
```

- Fetch time can be expensive. If many uneeded rows are likely to be returned, add '`TOP N`' to the query.

  > **NOTE:**
  > `TOP N` is applied only at the fetch time, after the full run is finished. It does not affect run time.

  Example

```
Select top 100 col1, col2, col10
from my_table
where col7='my_predicate';
```

- Functions on integers return a rounded integer.
  For mathematical operations on integers, use either use `CAST TO FLOAT` or a decimal number.

  Example

```
Calculate how much is 77% out of 500:

Select cast(500 as float)/77*100 ;
Or
Select 500/77.0*100;
```

- For large queries (multiple joins etc), consider using saved queries to save compilation time. Note that saved queries can use variables. See SQream SQL Reference Guide for saved queries examples, usage and limitations.