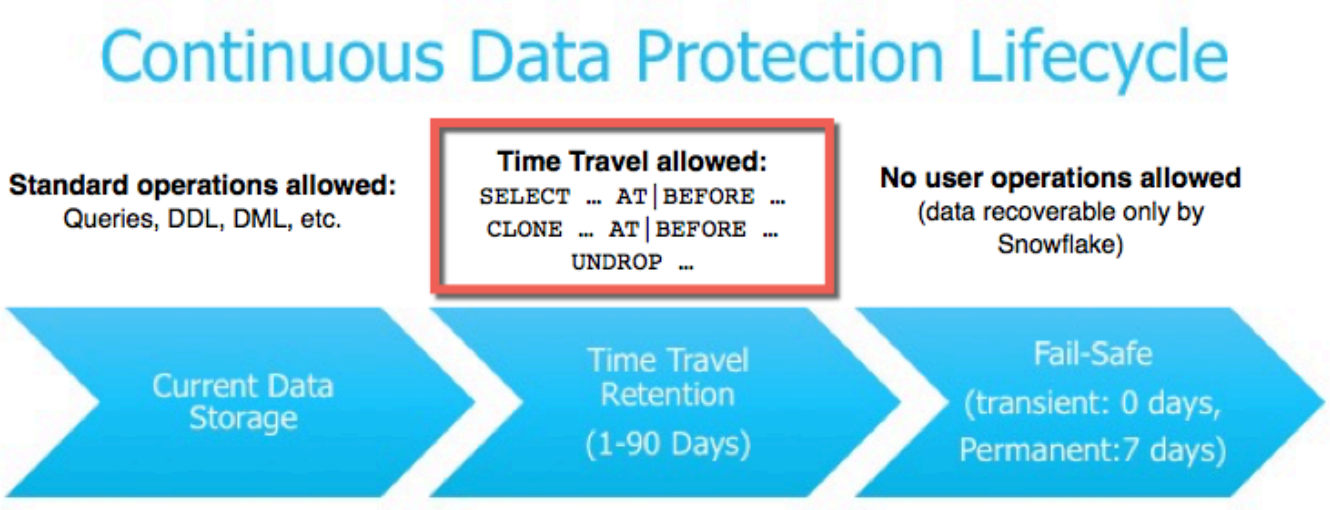
Understanding & using Time Travel

Snowflake Time Travel enables accessing historical data (i.e. data that has been changed or deleted) at any point within a defined period. It serves as a powerful tool for performing the following tasks:

- Restoring data-related objects (tables, schemas, and databases) that might have been accidentally or intentionally deleted.
- Duplicating and backing up data from key points in the past.
- Analyzing data usage/manipulation over specified periods of time.

Introduction to Time Travel



Using Time Travel, you can perform the following actions within a defined period of time:

- Query data in the past that has since been updated or deleted.
- Create clones of entire tables, schemas, and databases at or before specific points in the past.
- Restore tables, schemas, and databases that have been dropped.

Note
When querying historical data in a table or non-materialized view, the current table or view schema is used. For more information, see Usage notes for AT | BEFORE.

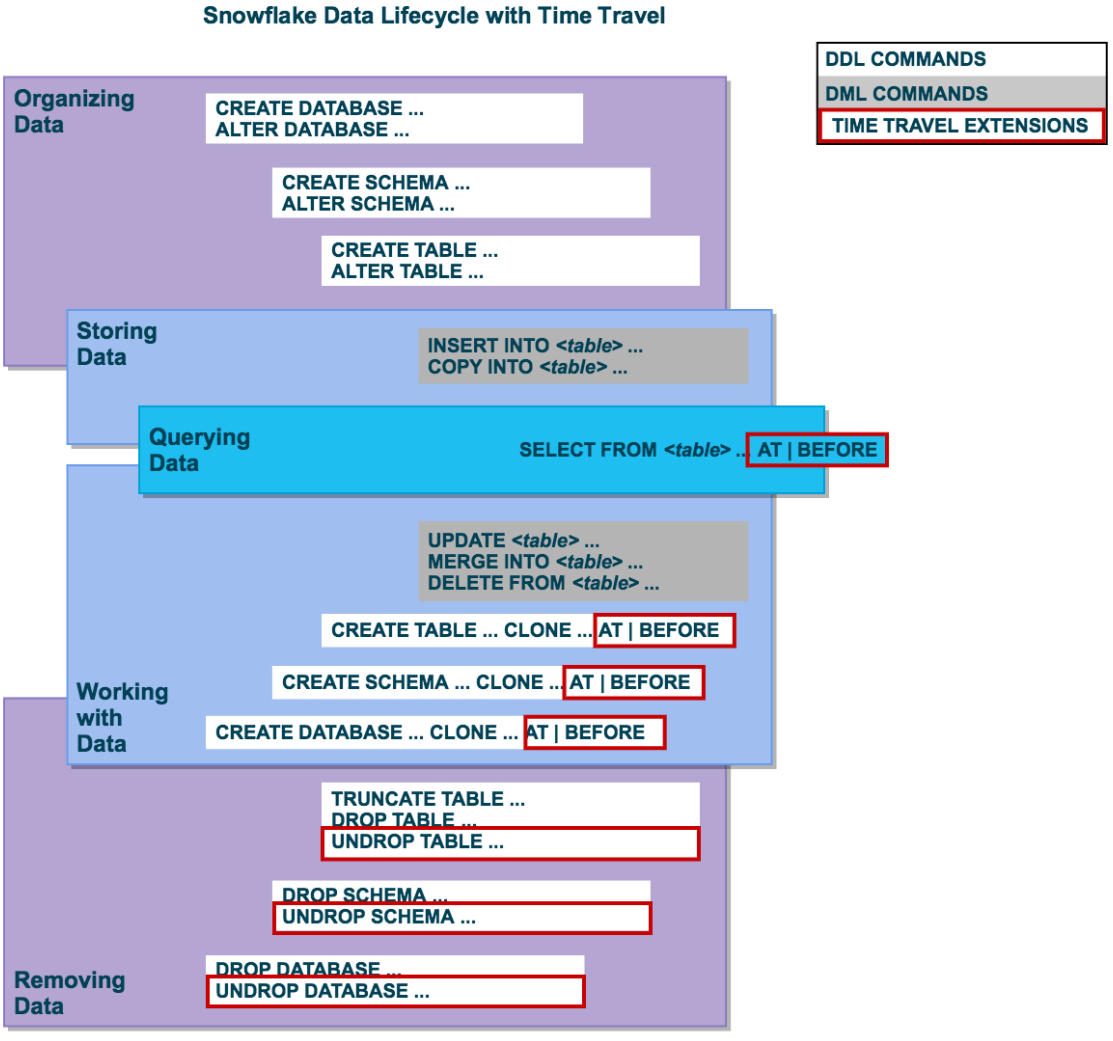
After the defined period of time has elapsed, the data is moved into Snowflake Fail-safe and these actions can no longer be performed.

> **Note**
> A long-running Time Travel query will delay moving any data and objects (tables, schemas, and databases) in the account into Fail-safe, until the query completes.

Time Travel SQL extensions

To support Time Travel, the following SQL extensions have been implemented:

- AT | BEFORE clause which can be specified in SELECT statements and CREATE ... CLONE commands (immediately after the object name). The clause uses one of the following parameters to pinpoint the exact historical data you want to access:
  - TIMESTAMP
  - OFFSET (time difference in seconds from the present time)
  - STATEMENT (query ID for statement)
- UNDROP command for tables, schemas, and databases.



Data retention period

A key component of Snowflake Time Travel is the data retention period.

When data in a table is modified, including deletion of data or dropping an object containing data, Snowflake preserves the state of the data before the update. The data retention period specifies the number of days for which this historical data is preserved and, therefore, Time Travel operations (SELECT, CREATE ... CLONE, UNDROP) can be performed on the data.

The standard retention period is 1 day (24 hours) and is automatically enabled for all Snowflake accounts:

- For Snowflake Standard Edition, the retention period can be set to 0 (or unset back to the default of 1 day) at the account and object level (i.e. databases, schemas, and tables).
- For Snowflake Enterprise Edition (and higher):
  - For transient databases, schemas, and tables, the retention period can be set to 0 (or unset back to the default of 1 day). The same is also true for temporary tables.
  - For permanent databases, schemas, and tables, the retention period can be set to any value from 0 up to 90 days.

> **Note**
> A retention period of 0 days for an object effectively deactivates Time Travel for the object.

When the retention period ends for an object, the historical data is moved into Snowflake Fail-safe:

- Historical data is no longer available for querying.
- Past objects can no longer be cloned.
- Past objects that were dropped can no longer be restored.

To specify the data retention period for Time Travel:

- The DATA_RETENTION_TIME_IN_DAYS object parameter can be used by users with the ACCOUNTADMIN role to set the default retention period for your account.
- The same parameter can be used to explicitly override the default when creating a database, schema, and individual table.
- The data retention period for a database, schema, or table can be changed at any time.
- The MIN_DATA_RETENTION_TIME_IN_DAYS account parameter can be set by users with the ACCOUNTADMIN role to set a minimum retention period for the account. This parameter does not alter or replace the DATA_RETENTION_TIME_IN_DAYS parameter value. However it may change the effective data retention time. When this parameter is set at the account

level, the effective minimum data retention period for an object is determined by MAX(DATA_RETENTION_TIME_IN_DAYS, MIN_DATA_RETENTION_TIME_IN_DAYS).

Enabling and deactivating Time Travel

No tasks are required to enable Time Travel. It is automatically enabled with the standard, 1-day retention period.

However, you may want to upgrade to Snowflake Enterprise Edition to enable configuring longer data retention periods of up to 90 days for databases, schemas, and tables. Note that extended data retention requires additional storage which will be reflected in your monthly storage charges. For more information about storage charges, see Storage Costs for Time Travel and Fail-safe.

Time Travel cannot be deactivated for an account. A user with the ACCOUNTADMIN role can set DATA_RETENTION_TIME_IN_DAYS to 0 at the account level, which means that all databases (and subsequently all schemas and tables) created in the account have no retention period by default; however, this default can be overridden at any time for any database, schema, or table.

A user with the ACCOUNTADMIN role can also set the MIN_DATA_RETENTION_TIME_IN_DAYS at the account level. This parameter setting enforces a minimum data retention period for databases, schemas, and tables. Setting MIN_DATA_RETENTION_TIME_IN_DAYS does not alter or replace the DATA_RETENTION_TIME_IN_DAYS parameter value. It may, however, change the effective data retention period for objects. When MIN_DATA_RETENTION_TIME_IN_DAYS is set at the account level, the data retention period for an object is determined by MAX(DATA_RETENTION_TIME_IN_DAYS, MIN_DATA_RETENTION_TIME_IN_DAYS).

Time Travel can be deactivated for individual databases, schemas, and tables by specifying DATA_RETENTION_TIME_IN_DAYS with a value of 0 for the object. However, if DATA_RETENTION_TIME_IN_DAYS is set to a value of 0, and MIN_DATA_RETENTION_TIME_IN_DAYS is set at the account level and is greater than 0, the higher value setting takes precedence.

Attention
Before setting DATA_RETENTION_TIME_IN_DAYS to 0 for any object, consider whether you want to deactivate Time Travel for the object, particularly as it pertains to recovering the object if it is dropped. When an object with no retention period is dropped, you will not be able to restore the object.

As a general rule, we recommend maintaining a value of (at least) 1 day for any given object.

If the Time Travel retention period is set to 0, any modified or deleted data is moved into Fail-safe (for permanent tables) or deleted (for transient tables) by a background process. This may take a short time to complete. During that time, the TIME_TRAVEL_BYTES in table storage metrics might contain a non-zero value even when the Time Travel retention period is 0 days.

Specifying the data retention period for an object

ENTERPRISE EDITION FEATURE

Specifying a retention period greater than 1 day requires Enterprise Edition (or higher). To inquire about upgrading, please contact Snowflake Support.

By default, the maximum retention period is 1 day (i.e. one 24 hour period). With Snowflake Enterprise Edition (and higher), the default for your account can be set to any value up to 90 days:

- When creating a table, schema, or database, the account default can be overridden using the DATA_RETENTION_TIME_IN_DAYS parameter in the command.
- If a retention period is specified for a database or schema, the period is inherited by default for all objects created in the database/schema.

A minimum retention period can be set on the account using the MIN_DATA_RETENTION_TIME_IN_DAYS parameter. If this parameter is set at the account level, the data retention period for an object is determined by MAX(DATA_RETENTION_TIME_IN_DAYS, MIN_DATA_RETENTION_TIME_IN_DAYS).

Changing the data retention period for an object

If you change the data retention period for a table, the new retention period impacts all data that is active, as well as any data currently in Time Travel. The impact depends on whether you increase or decrease the period:

Increasing Retention

Causes the data currently in Time Travel to be retained for the longer time period.

For example, if you have a table with a 10-day retention period and increase the period to 20 days, data that would have been removed after 10 days is now retained for an additional 10 days before moving into Fail-safe.

Note that this doesn't apply to any data that is older than 10 days and has already moved into Fail-safe.

Decreasing Retention

Reduces the amount of time data is retained in Time Travel:

- For active data modified after the retention period is reduced, the new shorter period applies.
- For data that is currently in Time Travel:
  - If the data is still within the new shorter period, it remains in Time Travel.
  - If the data is outside the new period, it moves into Fail-safe.

For example, if you have a table with a 10-day retention period and you decrease the period to 1-day, data from days 2 to 10 will be moved into Fail-safe, leaving only the data from day 1 accessible through Time Travel.

However, the process of moving the data from Time Travel into Fail-safe is performed by a background process, so the change is not immediately visible. Snowflake guarantees that the data will be moved, but does not specify when the process will complete; until the background process completes, the data is still accessible through Time Travel.

Note

If you change the data retention period for a database or schema, the change only affects active objects contained within the database or schema. Any objects that have been dropped (for example, tables) remain unaffected.

For example, if you have a schema `s1` with a 90-day retention period and table `t1` is in schema `s1`, table `t1` inherits the 90-day retention period. If you drop table `s1.t1`, `t1` is retained in Time Travel for 90 days. Later, if you change the schema's data retention period to 1 day, the retention period for the dropped table `t1` is unchanged. Table `t1` will still be retained in Time Travel for 90 days.

To alter the retention period of a dropped object, you must undrop the object, then alter its retention period.

To change the retention period for an object, use the appropriate ALTER <object> command. For example, to change the retention period for a table:

```
CREATE TABLE mytable(col1 NUMBER, col2 DATE) DATA_RETENTION_TIME_IN_DAYS=90;
```

```
ALTER TABLE mytable SET DATA_RETENTION_TIME_IN_DAYS=30;
```

Attention
Changing the retention period for your account or individual objects changes the value for all lower-level objects that do not have a retention period explicitly set. For example:

- If you change the retention period at the account level, all databases, schemas, and tables that do not have an explicit retention period automatically inherit the new retention period.
- If you change the retention period at the schema level, all tables in the schema that do not have an explicit retention period inherit the new retention period.

Keep this in mind when changing the retention period for your account or any objects in your account because the change might have Time Travel consequences that you did not anticipate or intend. In particular, we do ***not*** recommend changing the retention period to 0 at the account level.

Dropped containers and object retention inheritance

Currently, when a database is dropped, the data retention period for child schemas or tables, if explicitly set to be different from the retention of the database, is not honored. The child schemas or tables are retained for the same period of time as the database.

Similarly, when a schema is dropped, the data retention period for child tables, if explicitly set to be different from the retention of the schema, is not honored. The child tables are retained for the same period of time as the schema.

To honor the data retention period for these child objects (schemas or tables), drop them explicitly ***before*** you drop the database or schema.

Querying historical data

When any DML operations are performed on a table, Snowflake retains previous versions of the table data for a defined period of time. This enables querying earlier versions of the data using the AT | BEFORE clause.

This clause supports querying data either exactly at or immediately preceding a specified point in the table's history within the retention period. The specified point can be time-based (e.g. a timestamp or time offset from the present) or it can be the ID for a completed statement (e.g. SELECT or INSERT).

For example:

- The following query selects historical data from a table as of the date and time represented by the specified timestamp:

```
SELECT * FROM my_table AT(TIMESTAMP => 'Wed, 26 Jun 2024 09:20:00
-0700'::timestamp_tz);
```

- The following query selects historical data from a table as of 5 minutes ago:

```
SELECT * FROM my_table AT(OFFSET => -60*5);
```

- The following query selects historical data from a table up to, but not including any changes made by the specified statement:

```
SELECT * FROM my_table BEFORE(STATEMENT => '8e5d0ca9-005e-44e6-b858-
a8f5b37c5726');
```

Note
If the TIMESTAMP, OFFSET, or STATEMENT specified in the AT | BEFORE clause falls outside the data retention period for the table, the query fails and returns an error.

Cloning historical objects

In addition to queries, the AT | BEFORE clause can be used with the CLONE keyword in the CREATE command for a table, schema, or database to create a logical duplicate of the object at a specified point in the object's history.

For example:

- The following CREATE TABLE statement creates a clone of a table as of the date and time represented by the specified timestamp:

```
CREATE TABLE restored_table CLONE my_table
  AT(TIMESTAMP => 'Wed, 26 Jun 2024 01:01:00 +0300'::timestamp_tz);
```

- The following CREATE SCHEMA statement creates a clone of a schema and all its objects as they existed 1 hour before the current time:

```
CREATE SCHEMA restored_schema CLONE my_schema AT(OFFSET => -3600);
```

- The following CREATE DATABASE statement creates a clone of a database and all its objects as they existed prior to the completion of the specified statement:

```
CREATE DATABASE restored_db CLONE my_db
  BEFORE(STATEMENT => '8e5d0ca9-005e-44e6-b858-a8f5b37c5726');
```

Note
The cloning operation for a database or schema fails:

- If the specified Time Travel time is beyond the retention time of ***any current child*** (e.g., a table) of the entity.

  As a workaround for child objects that have been purged from Time Travel, use the IGNORE TABLES WITH INSUFFICIENT DATA RETENTION parameter of the CREATE <object> ... CLONE command. For more information, see Child objects and data retention time.
- If the specified Time Travel time is at or before the point in time when the object was created.

- The following CREATE DATABASE statement creates a clone of a database and all its objects as they existed four days ago, skipping any tables that have a data retention period of less than four days:

```
CREATE DATABASE restored_db CLONE my_db
  AT(TIMESTAMP => DATEADD(days, -4, current_timestamp)::timestamp_tz)
  IGNORE TABLES WITH INSUFFICIENT DATA RETENTION;
```

Dropping and restoring objects

Dropping objects

When a table, schema, or database is dropped, it is not immediately overwritten or removed from the system. Instead, it is retained for the data retention period for the object, during which time the object can be restored. Once dropped objects are moved to Fail-safe, you cannot restore them.

To drop a table, schema, or database, use the following commands:

- DROP TABLE
- DROP SCHEMA
- DROP DATABASE

> **Note**
> After dropping an object, creating an object with the same name does not restore the object. Instead, it creates a new version of the object. The original, dropped version is still available and can be restored.
>
> Restoring a dropped object restores the object in place (i.e. it does not create a new object).

Listing dropped objects

Dropped tables, schemas, and databases can be listed using the following commands with the HISTORY keyword specified:

- SHOW TABLES
- SHOW SCHEMAS
- SHOW DATABASES

For example:

```
SHOW TABLES HISTORY LIKE 'load%' IN mytestdb.myschema;

SHOW SCHEMAS HISTORY IN mytestdb;

SHOW DATABASES HISTORY;
```

The output includes all dropped objects and an additional DROPPED_ON column, which displays the date and time when the object was dropped. If an object has been dropped more than once, each version of the object is included as a separate row in the output.

> **Note**
> After the retention period for an object has passed and the object has been purged, it is no longer displayed in the SHOW *<object_type>* HISTORY output.

Restoring objects

A dropped object that has not been purged from the system (i.e. the object is displayed in the SHOW *<object_type>* HISTORY output) can be restored using the following commands:

- UNDROP TABLE
- UNDROP SCHEMA
- UNDROP DATABASE

Calling UNDROP restores the object to its most recent state before the DROP command was issued.

For example:

```
UNDROP TABLE mytable;

UNDROP SCHEMA myschema;

UNDROP DATABASE mydatabase;
```

Note
If an object with the same name already exists, UNDROP fails. You must rename the existing object, which then enables you to restore the previous version of the object.

Access control requirements and name resolution

Similar to dropping an object, a user must have OWNERSHIP privileges for an object to restore it. In addition, the user must have CREATE privileges on the object type for the database or schema where the dropped object will be restored.

Restoring tables and schemas is only supported in the current schema or current database, even if a fully-qualified object name is specified.

Example: Dropping and restoring a table multiple times

In the following example, the `mytestdb.public` schema contains two tables: `loaddata1` and `proddata1`. The `loaddata1` table is dropped and recreated twice, creating three versions of the table:

- Current version

- Second (i.e. most recent) dropped version
- First dropped version

The example then illustrates how to restore the two dropped versions of the table:

1. First, the current table with the same name is renamed to `loaddata3`. This enables restoring the most recent version of the dropped table, based on the timestamp.
2. Then, the most recent dropped version of the table is restored.
3. The restored table is renamed to `loaddata2` to enable restoring the first version of the dropped table.
4. Lastly, the first version of the dropped table is restored.

```
SHOW TABLES HISTORY;

+---------------------------------+-----------+---------------+-----------
--+-------+---------+------------+------+-------+--------+---------------
+---------------------------------+
| created_on                      | name      | database_name |
schema_name | kind  | comment | cluster_by | rows | bytes | owner  |
retention_time | dropped_on                     |
|---------------------------------+-----------+---------------+-----------
--+-------+---------+------------+------+-------+--------+---------------
+---------------------------------|
| Tue, 17 Mar 2016 17:41:55 -0700 | LOADDATA1 | MYTESTDB      | PUBLIC
| TABLE |         |            | 48   | 16248 | PUBLIC | 1              |
[NULL]                          |
| Tue, 17 Mar 2016 17:51:30 -0700 | PRODDATA1 | MYTESTDB      | PUBLIC
| TABLE |         |            | 12   | 4096  | PUBLIC | 1              |
[NULL]                          |
+---------------------------------+-----------+---------------+-----------
--+-------+---------+------------+------+-------+--------+---------------
+---------------------------------+

DROP TABLE loaddata1;

SHOW TABLES HISTORY;

+---------------------------------+-----------+---------------+-----------
--+-------+---------+------------+------+-------+--------+---------------
+---------------------------------+
| created_on                      | name      | database_name |
schema_name | kind  | comment | cluster_by | rows | bytes | owner  |
retention_time | dropped_on                     |
|---------------------------------+-----------+---------------+-----------
--+-------+---------+------------+------+-------+--------+---------------
+---------------------------------|
| Tue, 17 Mar 2016 17:51:30 -0700 | PRODDATA1 | MYTESTDB      | PUBLIC
| TABLE |         |            | 12   | 4096  | PUBLIC | 1              |
[NULL]                          |
| Tue, 17 Mar 2016 17:41:55 -0700 | LOADDATA1 | MYTESTDB      | PUBLIC
| TABLE |         |            | 48   | 16248 | PUBLIC | 1              |
```

```
Fri, 13 May 2016 19:04:46 -0700 |
+---------------------------------+-----------+---------------+-----------
--+-------+---------+------------+------+-------+--------+---------------
+---------------------------------+

CREATE TABLE loaddata1 (c1 number);
INSERT INTO loaddata1 VALUES (1111), (2222), (3333), (4444);

DROP TABLE loaddata1;

CREATE TABLE loaddata1 (c1 varchar);

SHOW TABLES HISTORY;

+---------------------------------+-----------+---------------+-----------
--+-------+---------+------------+------+-------+--------+---------------
+---------------------------------+
| created_on                      | name      | database_name |
schema_name | kind  | comment | cluster_by | rows | bytes | owner  |
retention_time | dropped_on                     |
|---------------------------------+-----------+---------------+-----------
--+-------+---------+------------+------+-------+--------+---------------
+---------------------------------|
| Fri, 13 May 2016 19:06:01 -0700 | LOADDATA1 | MYTESTDB      | PUBLIC
| TABLE |         |            | 0    | 0     | PUBLIC | 1              |
[NULL]                          |
| Tue, 17 Mar 2016 17:51:30 -0700 | PRODDATA1 | MYTESTDB      | PUBLIC
| TABLE |         |            | 12   | 4096  | PUBLIC | 1              |
[NULL]                          |
| Fri, 13 May 2016 19:05:32 -0700 | LOADDATA1 | MYTESTDB      | PUBLIC
| TABLE |         |            | 4    | 4096  | PUBLIC | 1              |
Fri, 13 May 2016 19:05:51 -0700 |
| Tue, 17 Mar 2016 17:41:55 -0700 | LOADDATA1 | MYTESTDB      | PUBLIC
| TABLE |         |            | 48   | 16248 | PUBLIC | 1              |
Fri, 13 May 2016 19:04:46 -0700 |
+---------------------------------+-----------+---------------+-----------
--+-------+---------+------------+------+-------+--------+---------------
+---------------------------------+

ALTER TABLE loaddata1 RENAME TO loaddata3;

UNDROP TABLE loaddata1;

SHOW TABLES HISTORY;

+---------------------------------+-----------+---------------+-----------
--+-------+---------+------------+------+-------+--------+---------------
+---------------------------------+
| created_on                      | name      | database_name |
schema_name | kind  | comment | cluster_by | rows | bytes | owner  |
retention_time | dropped_on                     |
|---------------------------------+-----------+---------------+-----------
--+-------+---------+------------+------+-------+--------+---------------
+---------------------------------|
| Fri, 13 May 2016 19:05:32 -0700 | LOADDATA1 | MYTESTDB      | PUBLIC
| TABLE |         |            | 4    | 4096  | PUBLIC | 1              |
```

```
[NULL]                          |
| Fri, 13 May 2016 19:06:01 -0700 | LOADDATA3 | MYTESTDB      | PUBLIC
| TABLE |         |            | 0    | 0     | PUBLIC | 1              |
[NULL]                          |
| Tue, 17 Mar 2016 17:51:30 -0700 | PRODDATA1 | MYTESTDB      | PUBLIC
| TABLE |         |            | 12   | 4096  | PUBLIC | 1              |
[NULL]                          |
| Tue, 17 Mar 2016 17:41:55 -0700 | LOADDATA1 | MYTESTDB      | PUBLIC
| TABLE |         |            | 48   | 16248 | PUBLIC | 1              |
Fri, 13 May 2016 19:04:46 -0700 |
+---------------------------------+-----------+---------------+-----------
--+-------+---------+------------+------+-------+--------+---------------
+---------------------------------+

ALTER TABLE loaddata1 RENAME TO loaddata2;

UNDROP TABLE loaddata1;

+---------------------------------+-----------+---------------+-----------
--+-------+---------+------------+------+-------+--------+---------------
+---------------------------------+
| created_on                      | name      | database_name |
schema_name | kind  | comment | cluster_by | rows | bytes | owner  |
retention_time | dropped_on                     |
|---------------------------------+-----------+---------------+-----------
--+-------+---------+------------+------+-------+--------+---------------
+---------------------------------|
| Tue, 17 Mar 2016 17:41:55 -0700 | LOADDATA1 | MYTESTDB      | PUBLIC
| TABLE |         |            | 48   | 16248 | PUBLIC | 1              |
[NULL]                          |
| Fri, 13 May 2016 19:05:32 -0700 | LOADDATA2 | MYTESTDB      | PUBLIC
| TABLE |         |            | 4    | 4096  | PUBLIC | 1              |
[NULL]                          |
| Fri, 13 May 2016 19:06:01 -0700 | LOADDATA3 | MYTESTDB      | PUBLIC
| TABLE |         |            | 0    | 0     | PUBLIC | 1              |
[NULL]                          |
| Tue, 17 Mar 2016 17:51:30 -0700 | PRODDATA1 | MYTESTDB      | PUBLIC
| TABLE |         |            | 12   | 4096  | PUBLIC | 1              |
[NULL]                          |
+---------------------------------+-----------+---------------+-----------
--+-------+---------+------------+------+-------+--------+---------------
+---------------------------------+
```