Db2 Iceberg – Chapter 2: Time Travel Queries

Time travel is a unique feature of Iceberg, allowing for historic versions of a table to be queried, which is possible due to the snapshot-based architecture of Iceberg tables. As discussed in the first installment of this blog post series on Db2 Iceberg, called [Tip of the (Db2) Iceberg](https://www.idug.org/news/tip-of-the-db2-iceberg), each transactional operation on an Iceberg table creates a snapshot representing the state and contents of the table at that moment. Using time travel, we can reference these snapshots to query the table’s data at that point in time or even combine multiple time travel queries across different snapshots or tables for more complex operations across the history of these tables.

# Time Travel Queries

Let’s look at the syntax of a time travel query. The time travel period which specifies the point in time we wish to query, is provided as a clause on the target table reference. There are two ways to specify the time travel period, which provides some flexibility in how we can specify the target time period to query based on the requirements of the application.

**FOR SYSTEM\_VERSION AS OF <snapshot\_id or snapshot\_reference>**

The SYSTEM\_VERSION period allows us to specify a target snapshot to query. We can provide either the snapshot ID as an integer or the snapshot reference as a string to specify the target snapshot. The snapshot ID must be provided as an integer type to distinguish a snapshot ID from a snapshot reference that looks like an integer.

**FOR SYSTEM\_TIME AS OF <timestamp>**

The SYSTEM\_TIME period allows us to specify a timestamp to query the table at a point in time. The timestamp will be resolved to the latest snapshot created prior to the given timestamp. The timestamp must be provided as a type that can be cast to a timestamp.

Let’s look at some examples of how we can use time travel on a simple Iceberg table. First, let’s create and populate the table and create a named reference for the first snapshot:

$ db2 "CREATE DATALAKE TABLE EMPLOYEE (ID INT, NAME VARCHAR(20)) STORED BY ICEBERG LOCATION 'db2remote://odfdefault//samples/employee' TBLPROPERTIES ('external.table.purge'='true')"

DB20000I The SQL command completed successfully.

# Insert two rows to create the first snapshot

$ db2 "INSERT INTO EMPLOYEE VALUES (1, 'Sammy Bean'), (2, 'Sarah Richards')"

DB20000I The SQL command completed successfully.

# Create a tag to refer to the first (current) snapshot

$ db2 "CALL SYSHADOOP.CREATE\_TAG('SAMPLES', 'EMPLOYEE', 'tag1')"

Result set 1

--------------

STATUS\_CODE STATUS\_MESSAGE

----------- ---------------------------------------------------------------------------

0 Tag tag1 created for snapshot 9055205995967104387 on table SAMPLES.EMPLOYEE

1 record(s) selected.

Return Status = 0

# Insert two more rows to create the second snapshot

$ db2 "INSERT INTO EMPLOYEE VALUES (3, 'Amy Dean'), (4, 'Fred Rogers')"

DB20000I The SQL command completed successfully.

# Selecting from the table we can see the four rows currently in the table

$ db2 "SELECT \* FROM EMPLOYEE"

ID NAME

----------- --------------------

3 Amy Dean

4 Fred Rogers

1 Sammy Bean

2 Sarah Richards

4 record(s) selected.

We can view information on the table’s snapshots using the [TABLE\_SNAPSHOTS](https://www.ibm.com/docs/en/db2/12.1?topic=functions-table-snapshots-table-function) table function:

$ db2 "SELECT \* FROM TABLE(SYSHADOOP.TABLE\_SNAPSHOTS('SAMPLES', 'EMPLOYEE'))"

SCHEMA\_NAME TABLE\_NAME SEQNUM CREATE\_TS SNAPSHOT\_ID PARENT\_ID SCHEMA\_ID OPERATION IS\_CURRENT

-------------------- -------------------- -------------------- -------------------------- -------------------- -------------------- ----------- ---------------- ----------

SAMPLES EMPLOYEE 1 2025-03-04-13.05.15.240000 9055205995967104387 - 0 append N

SAMPLES EMPLOYEE 2 2025-03-04-13.06.16.765000 468173151043008168 9055205995967104387 0 append Y

2 record(s) selected.

And the table’s snapshot references using the [TABLE\_SNAPSHOT\_REFS](https://www.ibm.com/docs/en/db2/12.1?topic=functions-table-snapshot-refs-table-function) table function:

$ db2 "SELECT VARCHAR(NAME, 32) NAME, SNAPSHOT\_ID FROM TABLE(SYSHADOOP.TABLE\_SNAPSHOT\_REFS('SAMPLES', 'EMPLOYEE')) WHERE TYPE = 'TAG'"

NAME SNAPSHOT\_ID

-------------------------------- --------------------

tag1 9055205995967104387

1 record(s) selected.

To query our example table as of the first snapshot, we could use any of the following statements:

-- Specify the snapshot ID of the first snapshot

SELECT \* FROM EMPLOYEE FOR SYSTEM\_VERSION AS OF 9055205995967104387;

-- Specify the named reference to the first snapshot

SELECT \* FROM EMPLOYEE FOR SYSTEM\_VERSION AS OF 'tag1';

-- Specify a timestamp prior to the second snapshot's create timestamp

SELECT \* FROM EMPLOYEE FOR SYSTEM\_TIME AS OF '2025-03-04-13.06.00.000000';

All three of these methods will yield the same result:

$ db2 "SELECT \* FROM EMPLOYEE FOR SYSTEM\_VERSION AS OF 9055205995967104387"

ID NAME

----------- --------------------

1 Sammy Bean

2 Sarah Richards

2 record(s) selected.

If we try to query a non-existent snapshot ID or reference, or a timestamp that does not correspond to any snapshot, we will get an error that looks like the following:

$ db2 "SELECT \* FROM EMPLOYEE FOR SYSTEM\_VERSION AS OF 'non-existent-tag'"

ID NAME

----------- --------------------

SQL20524N The statement failed because of an invalid period specification or

period clause for period "SYSTEM\_VERSION". Reason code "10". SQLSTATE=428HY

*SQL20524N with reason code 10 will always mean that a snapshot could not be resolved for the given time travel period. For other reason codes please see the Db2 documentation for* [*SQL20524N*](https://www.ibm.com/docs/en/db2/12.1?topic=messages-sql20000-20999#sqlmsg__SQL20524N)*.*

## Alternatives for Specifying the Target Value

There is some flexibility in how we can specify the target snapshot or timestamp in the time travel clause. In addition to specifying the time travel target as literal values, we can also specify the target in a variety of ways such as using SQL variables or [special registers](https://www.ibm.com/docs/en/db2/12.1?topic=elements-special-registers). The restrictions on what can be used to specify the time travel target are as follows:

* The input must be a type that can be cast to a timestamp, except for integer types when specifying a snapshot ID
* No column references
* No sub-queries
* No user defined functions
* No non-deterministic functions

For example, if we wanted to provide ID of the first snapshot by querying the TABLE\_SNAPSHOTS table function, trying to do this using a sub-query will fail:

$ db2 "SELECT \* FROM EMPLOYEE FOR SYSTEM\_VERSION AS OF (SELECT SNAPSHOT\_ID FROM TABLE(SYSHADOOP.TABLE\_SNAPSHOTS('SAMPLES', 'EMPLOYEE')) WHERE SEQNUM = 1)"

SQL0104N An unexpected token "EMPLOYEE FOR SYSTEM\_VERSION AS OF" was

found following "SELECT \* FROM ". Expected tokens may include: "<space>".

SQLSTATE=42601

Instead, this can be achieved by using a SQL variable to store the result of the sub-query:

$ db2 "CREATE OR REPLACE VARIABLE TARGET\_SNAP BIGINT DEFAULT -1"

DB20000I The SQL command completed successfully.

$ db2 "SET TARGET\_SNAP = ((SELECT SNAPSHOT\_ID FROM TABLE(SYSHADOOP.TABLE\_SNAPSHOTS('SAMPLES', 'EMPLOYEE')) WHERE SEQNUM

= 1))"

DB20000I The SQL command completed successfully.

$ db2 "SELECT \* FROM EMPLOYEE FOR SYSTEM\_VERSION AS OF TARGET\_SNAP"

ID NAME

----------- --------------------

1 Sammy Bean

2 Sarah Richards

2 record(s) selected.

## Time Travel Behaviour involving Schema Changes

Continuing with our example table, let’s add a new column to this table and insert some new data to create a new snapshot:

$ db2 "ALTER DATALAKE TABLE EMPLOYEE ADD COLUMN AGE SMALLINT"

DB20000I The SQL command completed successfully.

$ db2 "INSERT INTO EMPLOYEE VALUES (5, 'John Smith', 34)"

DB20000I The SQL command completed successfully.

$ db2 "SELECT \* FROM EMPLOYEE"

ID NAME AGE

----------- -------------------- -----------

3 Amy Dean -

4 Fred Rogers -

5 John Smith 34

1 Sammy Bean -

2 Sarah Richards -

5 record(s) selected.

If we query the table at a point in time just before we added this column (you can use the TABLE\_SNAPSHOTS table function to help determine a timestamp to use), you’ll notice that the column still exists in the result set. Db2 will always use the table’s current schema, regardless of the schema originally used by the target snapshot.

$ db2 "SELECT \* FROM EMPLOYEE FOR SYSTEM\_TIME AS OF '2025-03-04-13.10.00.000000'"

ID NAME AGE

----------- -------------------- -----------

1 Sammy Bean -

2 Sarah Richards -

3 Amy Dean -

4 Fred Rogers -

4 record(s) selected.

## Querying Changes to a Table using Time Travel

Let’s look at a more complex example where we can leverage time travel. If we wanted to see what changes have been made to an Iceberg table between two points in time, we can query the table at each point in time and take the difference to see the changes. By adjusting the order in which we compare the table’s datasets, we can determine which rows were added and which were removed. Let’s first delete a couple rows from our example table:

$ db2 "DELETE FROM EMPLOYEE WHERE ID IN (2,3)"

DB20000I The SQL command completed successfully.

$ db2 "CALL SYSHADOOP.CREATE\_TAG('SAMPLES', 'EMPLOYEE', 'tag4')"

Result set 1

--------------

STATUS\_CODE STATUS\_MESSAGE

----------- ---------------------------------------------------------------------------

0 Tag tag4 created for snapshot 6490697599557248908 on table SAMPLES.EMPLOYEE

1 record(s) selected.

Return Status = 0

$ db2 "SELECT \* FROM EMPLOYEE"

ID NAME AGE

----------- -------------------- -----------

5 John Smith 34

4 Fred Rogers -

1 Sammy Bean -

3 record(s) selected.

We can use a query like the following to outline the rows that were added or removed between the first and fourth snapshots (referenced by ‘tag1’ and ‘tag4’ respectively):

SELECT i.\*, 'INSERTED' action FROM

TABLE(

-- determine which rows were inserted across the snapshots

SELECT \* FROM EMPLOYEE FOR SYSTEM\_VERSION AS OF 'tag4'

EXCEPT

SELECT \* FROM EMPLOYEE FOR SYSTEM\_VERSION AS OF 'tag1'

) i

UNION ALL

SELECT d.\*, 'DELETED' action FROM

TABLE(

-- determine which rows were deleted across the snapshots

SELECT \* FROM EMPLOYEE FOR SYSTEM\_VERSION AS OF 'tag1'

EXCEPT

SELECT \* FROM EMPLOYEE FOR SYSTEM\_VERSION AS OF 'tag4'

) d;

Running this gives us the following result set on our example table *(note the absence of ID=3, which was added after ‘tag1’ but deleted before ‘tag4’)*:

ID NAME AGE ACTION

----------- -------------------- ----------- --------

2 Sarah Richards - DELETED

5 John Smith 34 INSERTED

4 Fred Rogers - INSERTED

3 record(s) selected.

# Snapshot Rollback

While time travel allows us to peek at historic versions of a table, we also have the ability to rollback or set a table’s current snapshot to one in the past. By setting the current snapshot to a different snapshot we can achieve a similar effect to time travel, however all operations on the table will create new snapshots stemming from the snapshot we rolled back to, creating a new fork in the table’s snapshot history. Additionally, rolling back to a previous snapshot will affect all other database engines’ view of this table, so this should only be done when we want to go back to a previous version of the table and not as an alternative form of time travel.

We can rollback to a snapshot (meaning the target snapshot must be an ancestor of the current snapshot) using the [ROLLBACK\_TO\_SNAPSHOT](https://www.ibm.com/docs/en/db2/12.1?topic=procedures-rollback-snapshot) or [ROLLBACK\_TO\_TIMESTAMP](https://www.ibm.com/docs/en/db2/12.1?topic=procedures-rollback-timestamp) stored procedures, or we can set the snapshot (no restriction on the relation of the target snapshot to the current snapshot) using the [SET\_TO\_SNAPSHOT](https://www.ibm.com/docs/en/db2/12.1?topic=procedures-set-snapshot) or [SET\_TO\_TIMESTAMP](https://www.ibm.com/docs/en/db2/12.1?topic=procedures-set-timestamp) stored procedures. Similar to the behaviour previously mentioned for time travel, the table schema will not change to the schema used by the target snapshot.

Looking at our example table, we can rollback to the first snapshot as follows:

$ db2 "CALL SYSHADOOP.ROLLBACK\_TO\_SNAPSHOT('SAMPLES', 'EMPLOYEE', 9055205995967104387)"

Result set 1

--------------

STATUS\_CODE STATUS\_MESSAGE

----------- ----------------------------------------------------------------------------------------------

0 SAMPLES.EMPLOYEE rolled back to snapshot 9055205995967104387 created at 2025-03-04 13:05:15.24

1 record(s) selected.

Return Status = 0

$ db2 "SELECT \* FROM EMPLOYEE"

ID NAME AGE

----------- -------------------- -----------

1 Sammy Bean -

2 Sarah Richards -

2 record(s) selected.

Since rolling back only changes the current snapshot, we can still use time travel and leverage the special register [CURRENT TIMESTAMP](https://www.ibm.com/docs/en/db2/12.1?topic=registers-current-timestamp) to query the most recent version of the table without needing to first determine the latest snapshot:

$ db2 "SELECT \* FROM EMPLOYEE FOR SYSTEM\_TIME AS OF CURRENT TIMESTAMP"

ID NAME AGE

----------- -------------------- -----------

3 Amy Dean -

4 Fred Rogers -

5 John Smith 34

1 Sammy Bean -

2 Sarah Richards -

5 record(s) selected.

*There are cases where using CURRENT TIMESTAMP or CURRENT DATE may fail to resolve to a snapshot. These special registers return the time-of-day of the current Db2 server, which may differ from the time-of-day of the server which wrote the snapshots.*

# Differences with Db2 Temporal Tables

Db2 users who are familiar with [time travel on Db2 temporal tables](https://www.ibm.com/docs/en/db2/12.1?topic=tables-time-travel-query-using-temporal) may recognize the similar syntax used for Iceberg time travel, particularly when using the SYSTEM\_TIME period. However, despite sharing some syntax Iceberg time travel has many differences when compared to Db2 temporal tables:

* Queries using Iceberg time travel functionality read the data files associated with the snapshot referenced, whereas time travel queries using Db2 temporal tables read the data that existed or was valid for a given time period based on special timestamp columns in the table.
* Iceberg time travel is baked into the design of Iceberg tables, where Db2 temporal tables must be created with these specific timestamp rows and correlated to a history table in some cases.
* The period used for Iceberg time travel refers to how we should resolve the target snapshot based on the supplied input, where the period in Db2 temporal tables has different semantic meanings: System-period tables track when rows were created, updated, and deleted, and application-period tables track when a row is considered to be valid.
  + Iceberg time travel also introduces the SYSTEM\_VERSION period, which has no application for Db2 temporal tables and is not supported.
* Iceberg time travel only supports the AS OF range, where time travel using Db2 temporal tables additionally support the BETWEEN and FROM .. TO ranges.
* The [CURRENT TEMPORAL SYSTEM\_TIME](https://www.ibm.com/docs/en/db2/12.1?topic=registers-current-temporal-system-time) and [CURRENT TEMPORAL BUSINESS\_TIME](https://www.ibm.com/docs/en/db2/12.1?topic=registers-current-temporal-business-time) special registers have no implicit effect on Iceberg time travel.
  + Iceberg time travel can still reference the value of the special register, i.e. FOR SYSTEM\_TIME AS OF CURRENT TEMPORAL SYSTEM\_TIME

# Additional Resources

* <https://www.ibm.com/docs/en/db2/12.1?topic=tables-iceberg-time-travel-queries>
* <https://iceberg.apache.org/spec/#point-in-time-reads-time-travel>