Practice: Using Blockchain Tables

Practice Target

The goal of this practice is to familiarize you with creating and managing Blockchain Tables in Oracle Database 23ai. You will learn how to establish a tamper-evident ledger for transactional data, and enforce retention policies.

Practice Overview

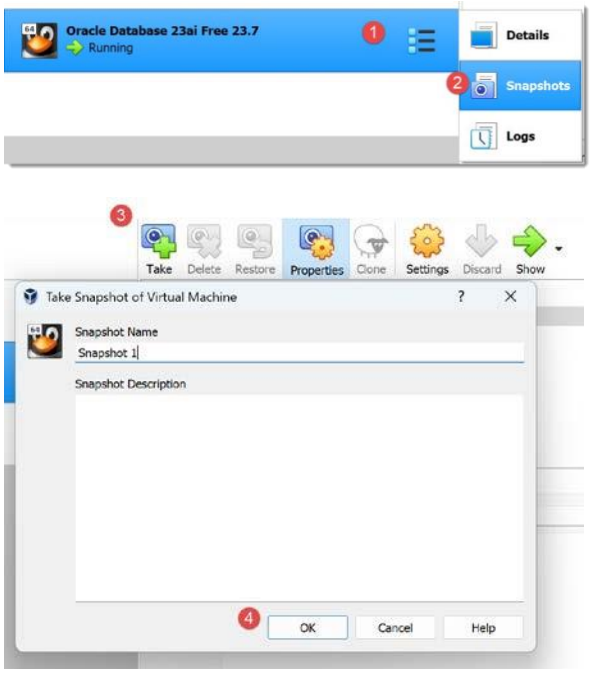
In high level, you will perform the following tasks:

1. **Create a Blockchain Table**
   Set up a blockchain table to store transaction records.
2. **Updating and Altering a Blockchain Table**
   Attempt to update, truncate, drop, or alter the table to see how certain DML and DDL operations are restricted.
3. **Add/Drop Columns**
   Learn how to modify the table structure safely within blockchain constraints.
4. **User Chains**
   Create custom user-defined cryptographic chains for specific columns.
5. **Row Versions**
   Explore how row versioning works in a blockchain table for auditing.
6. **Control Long Idle Retention Times**
   Experiment with retention policies to prevent or allow dropping a table after certain idle periods.

Creating a Blockchain Table

In this first section, you will create a simple blockchain table to store transaction data.

1. In Oracle VirtualBox manager, take a snapshot of the vm.



Caution: Do not proceed with the practice before taking a snapshot of the vm. 2.

Start **SQL Developer** and connect as **SOE** to the pdb

3. Create a new blockchain table:

```
CREATE BLOCKCHAIN TABLE bc_txn_log (
txn_id       NUMBER PRIMARY KEY,
account_no   NUMBER,
txn_amount   NUMBER,
txn_date     DATE
)
NO DROP UNTIL 0 DAYS IDLE
NO DELETE UNTIL 16 DAYS AFTER INSERT
HASHING USING "SHA2_512"
VERSION V2;
```

Note: In Oracle 23ai, you can specify different retention rules, such as shorter or longer *idle* or *after insert* periods, depending on your requirements. In this example, the retention period is set to zero so that we can drop the table after using it for demonstration purposes.

4. Check the invisible columns created in the table.

The meaning of the hidden columns are described in the documentation.

```
SELECT   internal_column_id, column_name, data_type, data_length, hidden_column
```

```
FROM     user_tab_cols
WHERE    table_name = 'BC_TXN_LOG' ORDER BY internal_column_id;
```

5. Query the view USER_BLOCKCHAIN_TABLES. It displays information about blockchain tables.

```
select table_name,
row_retention,
row_retention_locked,
table_inactivity_retention,
hash_algorithm,        table_version
from   user_blockchain_tables  where
table_name like 'BC_TXN_LOG%' order
by 1;
```

Updating and Altering a Blockchain Table

Blockchain tables are designed to prevent unauthorized modifications. In this section, you'll see how certain DML and DDL operations are either restricted.

6. Try to drop the table.

   The drop operation succeeds and did not obey the drop retention period because the table is empty.

```
DROP TABLE IF EXISTS bc_txn_log purge;
```

7. Recreate the table with zero drop retention period.

```
CREATE BLOCKCHAIN TABLE bc_txn_log (     txn_id       NUMBER PRIMARY KEY,
account_no   NUMBER,     txn_amount   NUMBER,     txn_date     DATE
)
NO DROP UNTIL 0 DAYS IDLE
NO DELETE UNTIL 16 DAYS AFTER INSERT
HASHING USING "SHA2_512" VERSION V2;
```

8. Attempt to insert data into the blockchain table:

```
INSERT INTO bc_txn_log (txn_id, account_no, txn_amount, txn_date)
VALUES (1, 12345, 500, SYSDATE);

COMMIT;  -- Required to finalize the insert
```

9. Now try an UPDATE or DELETE on a row to see what happens:

```
UPDATE bc_txn_log SET txn_amount = 600 WHERE txn_id = 1;
DELETE bc_txn_log WHERE txn_id = 1;
```

You will typically receive an error (e.g., "ORA-... operation not supported on blockchain table") because updates are not allowed.

10. Attempt a **TRUNCATE** operation within the restricted retention period:

```
TRUNCATE TABLE bc_txn_log;  -- Should fail or be disallowed
```

These DDL statements are blocked until the specified retention criteria is met (e.g., "16 days idle").

11. Increase the row retention to 32 days then try to reduce it back to 16 days.

Assuming the table wasn't defined as locked, you can modify the NO DELETE clause using an ALTER TABLE command, provided you do not shorten its retention period.

```
ALTER TABLE bc_txn_log NO DELETE UNTIL 32 DAYS AFTER INSERT;
ALTER TABLE bc_txn_log NO DELETE UNTIL 16 DAYS AFTER INSERT;
```

Note: We can set the row retention to NO DELETE, which means rows will be kept forever using the command ALTER TABLE `bc_txn_log` NO DELETE;

Add/Drop Columns

As you would expect for an insert-only table, all DML and DDL operations that would result in row data being amended or deleted are prevented for a blockchain table.

12. Try adding a new column:

```
ALTER TABLE bc_txn_log
  ADD (region VARCHAR2(50));
```

13. Check if you can extend the column

```
ALTER TABLE bc_txn_log MODIFY (region VARCHAR2(51));
```

14. Check if you can drop the column

    The dropped columns are marked as hidden, rather than actually being dropped.

```
ALTER TABLE bc_txn_log DROP COLUMN region;
```

User Chains

In earlier releases, blockchain tables supported up to 32 system-generated chains per instance, and rows were assigned to these chains at random. In Oracle Database 23ai, we can now create **user chains**, using up to three columns to define each chain. Any unique combination of values for these columns corresponds to its own chain.

15. Re-create the blockchain table using a user-defined chain.

```
DROP TABLE bc_txn_log PURGE;

CREATE BLOCKCHAIN TABLE bc_txn_log (     txn_id       NUMBER PRIMARY KEY,
account_no   NUMBER,     txn_amount   NUMBER,     txn_date     DATE,     region
VARCHAR2(20)
)
NO DROP UNTIL 0 DAYS IDLE
NO DELETE UNTIL 16 DAYS AFTER INSERT
HASHING USING "SHA2_512"
WITH USER CHAIN reg_chain (region) VERSION V2;
```

16. Insert sample data into the bc_txn_log table, each having different region values.

```
INSERT INTO bc_txn_log (txn_id, account_no, region, txn_amount, txn_date)
VALUES (1, 1001, 'EMEA',     500, SYSDATE),
       (2, 1002, 'APAC',     750, SYSDATE),
       (3, 1003, 'AMERICAS', 300, SYSDATE),
       (4, 1004, 'APAC',     900, SYSDATE);

COMMIT;
```

17. Query the USER_BLOCKCHAIN_TABLE_CHAINS view to see the new chains created in your table.

```
SELECT table_name, chain_id
  FROM user_blockchain_table_chains
 WHERE table_name = 'BC_TXN_LOG';
```

You should see one user chain for the region column.

Row Versions

By default, rows in a blockchain table cannot be updated; each modification must be done by inserting a new row. Enabling **row versions** allows Oracle to track the sequence of those inserts over time and keep only the latest version for any set of related columns. To do this, you use the WITH ROW VERSION [AND USER CHAIN] clause when creating the blockchain table, specifying up to three columns that define how rows are related.

In the following example, we demonstrate *row versions* on the bc_txn_log table, basing the user chain on the account_no column.

18. Drop the table if it already exists, then create the blockchain table with WITH ROW VERSION AND USER CHAIN.

```
DROP TABLE bc_txn_log PURGE;

CREATE BLOCKCHAIN TABLE bc_txn_log
(     txn_id       NUMBER,
account_no   NUMBER,     txn_amount
NUMBER,     txn_date     DATE,
    CONSTRAINT bc_txn_log_pk PRIMARY KEY (txn_id)
)
NO DROP UNTIL 0 DAYS IDLE
NO DELETE UNTIL 16 DAYS AFTER INSERT
HASHING USING "SHA2_512"
WITH ROW VERSION AND USER CHAIN acct_chain (account_no) VERSION
"v2";
```

The acct_chain (account_no) clause ensures rows sharing the same account_no are treated as versions of the same chain. Each new row with the same account_no is considered a new version.

19. Insert sample data to demonstrate row versions. Notice how some rows share the same account_no but a different txn_id and txn_amount.

```
INSERT INTO bc_txn_log (txn_id, account_no, txn_amount, txn_date)
VALUES (1, 11111, 100, SYSDATE),
       (2, 22222, 200, SYSDATE),
       (3, 11111, 150, SYSDATE),
       (4, 22222, 250, SYSDATE);

COMMIT;
```

20. Query the entire bc_txn_log table to see all versions that were inserted:

```
SELECT * FROM bc_txn_log;
```

You should see multiple rows for accounts 11111 and 22222, representing each step in the version history for that account.

21. Because we specified WITH ROW VERSION AND USER CHAIN, Oracle creates an additional view named bc_txn_log_last$. Querying it returns only the latest row version for each unique account_no:

```
SELECT * FROM bc_txn_log_last$;
```

This allows you to view the most recent status of each account without manually filtering through older versions.

Control Long Idle Retenon Times

In Oracle Database 23ai, the maximum *idle* retention time you can set for a blockchain table is controlled by the BLOCKCHAIN_TABLE_RETENTION_THRESHOLD parameter. Attempting to exceed this value produces an error unless you either increase the parameter or are granted the necessary privilege to bypass it.

22. Connect as a privileged user (**SYS**) and check the current threshold value

```
SHOW PARAMETER blockchain_table_retention_threshold;
```

If it is set to 16, you cannot specify an idle retention greater than 16 days unless you raise this setting or have the TABLE RETENTION privilege.

23. Attempt to create a blockchain table named bc_txn_log with an idle retention of 32 days:

```
DROP TABLE bc_txn_log PURGE;

CREATE BLOCKCHAIN TABLE bc_txn_log (     txn_id       NUMBER PRIMARY KEY,
account_no   NUMBER,     region       VARCHAR2(50),     txn_amount   NUMBER,
txn_date     DATE
)
NO DROP UNTIL 32 DAYS IDLE
NO DELETE UNTIL 16 DAYS AFTER INSERT
HASHING USING "SHA2_512"
VERSION "v2";
```

Because the threshold is currently 16, you will receive an error like:

```
ORA-05807: Blockchain or immutable table "..." cannot have idle retention greater
than 16 days.
```

24. As **SYS** user, grant the TABLE RETENTION privilege to the user SOE:

```
GRANT TABLE RETENTION TO soe;
```

25. Connect again as **soe** and retry the table creation:

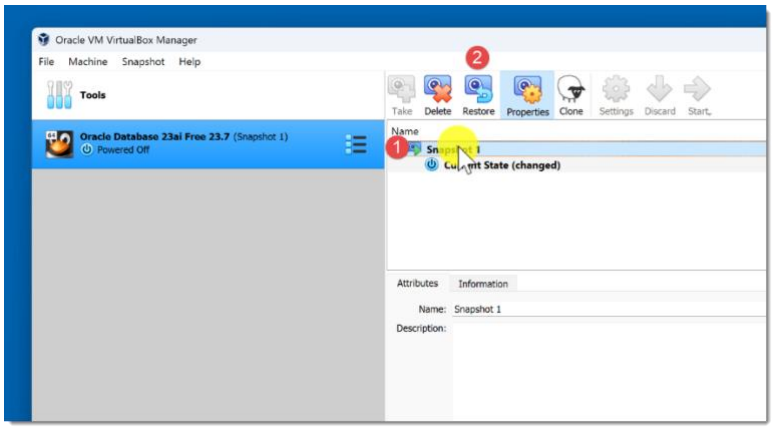
```
CREATE BLOCKCHAIN TABLE bc_txn_log
(     txn_id       NUMBER PRIMARY
KEY,     account_no   NUMBER,
region       VARCHAR2(50),
txn_amount   NUMBER,
    txn_date     DATE
)
```

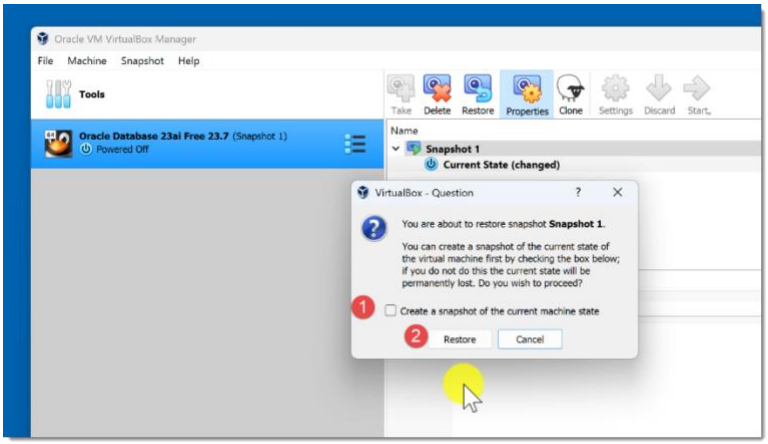
```
NO DROP UNTIL 32 DAYS IDLE
NO DELETE UNTIL 16 DAYS AFTER INSERT
HASHING USING "SHA2_512"
VERSION "v2";

-- Table created successfully now.
```

With the TABLE RETENTION privilege, the user can ignore the default blockchain_table_retention_threshold limit and set an idle retention period greater than 16 days.

Cleanup

26. Exit from SQL Developer.
27. Shutdown the vm
28. In Oracle VirtualBox manager, restore the vm from the snapshot taken in the beginning of the lecture.





29. Delete the snapshot.