43
DBMS_BLOCKCHAIN_TABLE

A blockchain table is an append-only table designed for centralized blockchain applications. The `DBMS_BLOCKCHAIN_TABLE` package allows you do operations like the following: delete rows in a blockchain table that are beyond the row retention defined for the blockchain table; get the bytes that are input to the cryptographic hash for a row so you can verify the hash in the row; sign a row you inserted into a blockchain table after the row is added to a chain in the blockchain table; and have the database verify the hashes and digital signatures on some or all rows in a blockchain table. V2 blockchain tables support schema evolution, delegate signatures, and countersignatures in addition to the functionality found in V1 blockchain tables. Blockchain tables support only `DER` encoding for `X.509` certificates, not `PEM` encoding.

This chapter contains the following topics:

See Also:

- *Oracle Database Administrator's Guide*
- *Oracle Database Concepts*
- *Oracle Database SQL Language Reference*
- *Oracle Database Reference*
- For information on hidden columns in blockchain tables, see Hidden Columns in Blockchain Tables

DBMS_BLOCKCHAIN_TABLE Overview

In Oracle Blockchain Table, peers are database users who trust the database to maintain a tamper-resistant ledger.

The ledger is implemented as a blockchain table, which is defined and managed by the application. Existing applications can protect against fraud without requiring a new infrastructure or programming model. Although transaction throughput is lower than for a standard table, performance for a blockchain table is better than for a decentralized blockchain.

The `DBMS_BLOCKCHAIN_TABLE` package lets you do the following:

- delete rows in a blockchain table that are beyond the row retention defined for the blockchain table
- get the bytes that are input to the signature algorithm so you can sign a row you inserted into the blockchain table

- get the bytes that are input to the cryptographic hash for a row so you can verify the hash in the row
- sign a row you inserted into a blockchain table after the row is added to a chain in the blockchain table
- have the database verify the hashes and signatures on some or all rows in a blockchain table
- enable a delegate with sufficient privileges to sign a row inserted by another user
- procure a countersignature for the row that is being signed by the end-user or delegate

DBMS_BLOCKCHAIN_TABLE Security Model

The `DBMS_BLOCKCHAIN_TABLE` package is owned by `SYS` and is installed as part of database installation. The routines in the package are run with invoker's rights (run with the privileges of the current user). Thus any user with select privileges on the blockchain table should be able to validate the row contents of that table.

Any user with delete privileges on the blockchain table can delete rows beyond the retention period defined for the blockchain table.

A user that inserted a row into the blockchain table can add a digital signature to the row after the row is added to a chain in the blockchain table. This user can secure a countersignature on this row either when signing the row or at a later time after the row has been signed either by the user or by a delegate.

A delegate signer for a row in a blockchain table needs the `SIGN` privilege on the table. In addition, the ID number of the delegate user may be specified in the hidden column `ORABCTAB_DELEGATE_USER_NUMBER$` when the row is inserted.

A user with the `SIGN` privilege on a blockchain table can secure a countersignature on any row in the blockchain table that has a user signature or a delegate signature. Similarly, the owner of a blockchain table can secure a countersignature on any row in the blockchain table that has a user signature or a delegate signature.

Summary of DBMS_BLOCKCHAIN_TABLE Subprograms

The `DBMS_BLOCKCHAIN_TABLE` package uses `ADD_INTERVAL_PARTITIONING`, `COUNTERSIGN_ROW`, `COUNTERSIGN_ROW_SPECIFIED_BY_KEY_COLUMNS`, `DELETE_EXPIRED_ROWS`, `GET_BLOCKCHAIN_DIGEST`, `GET_BLOCKCHAIN_DIGEST_FOR_SELECTED_ROWS`, `GET_BYTES_FOR_ROW_HASH`, `GET_BYTES_FOR_ROW_HASH_SPECIFIED_BY_KEY_COLUMNS`, `GET_BYTES_FOR_ROW_SIGNATURE`, `GET_BYTES_FOR_ROW_SIGNATURE_SPECIFIED_BY_KEY_COLUMNS`, `GET_SIGNED_BLOCKCHAIN_DIGEST`, `GET_SIGNED_BLOCKCHAIN_DIGEST_FOR SELECTED_ROWS`, `SIGN_ROW`, `SIGN_ROW_SPECIFIED_BY_KEY_COLUMNS`, `SIGN_ROW_SPECIFIED_BY_KEY_COLUMNS_WITH_COUNTERSIGNATURE`, `SIGN_ROW_WITH_COUNTERSIGNATURE`, `VERIFY_ROWS`, `VERIFY_TABLE_BLOCKCHAIN`, and `VERIFY_USER_BLOCKCHAIN_ROWS` subprograms to perform various functions.

Table 43-1 DBMS_BLOCKCHAIN_TABLE Package Subprograms

Subprogram	Description
ADD_INTERVAL_PARTITIONING Procedure	This procedure adds interval partitioning to an existing, non-partitioned, V1 or V2 blockchain table.

Table 43-1 (Cont.) DBMS_BLOCKCHAIN_TABLE Package Subprograms

Subprogram	Description
COUNTERSIGN_ROW Procedure	This procedure procures a countersignature on a specified row in a blockchain table. The countersignature will be produced by signing the row data content using the table owner's private key stored in the database wallet.
COUNTERSIGN_ROW_SPECIFIED_BY_KEY_COLUMNS Procedure	It is an extension of `COUNTERSIGN_ROW` that uses at most three user columns to identify exactly one row in a blockchain table and procure a countersignature on that row.
DELETE_EXPIRED_ROWS Procedure	This procedure deletes rows outside the retention window created before `before_timestamp` if the timestamp is specified; otherwise, this procedure deletes all rows outside the retention window. This procedure commits before deleting any expired rows and commits after deleting any expired rows.
GET_BLOCKCHAIN_DIGEST Function	This function generates and returns a cryptographic hash of the digest for a specified blockchain table.
GET_BLOCKCHAIN_DIGEST_FOR_SELECTED_ROWS Function	This function generates and returns a cryptographic hash of the digest for user-specified rows in a blockchain table.
GET_BYTES_FOR_ROW_HASH Procedure	This procedure returns in `row_data` the bytes for the particular row identified (a series of meta-data-value, column-data-value pairs in column position order) followed by the hash for the previous row in the chain in the data format specified.
GET_BYTES_FOR_ROW_HASH_SPECIFIED_BY_KEY_COLUMNS Procedure	It is an extension of `GET_BYTES_FOR_ROW_HASH` that uses at most three user columns instead of the `instance identifier`, `chain identifier`, and `sequence number` to uniquely identify the row.
GET_BYTES_FOR_ROW_SIGNATURE Procedure	This procedure returns the bytes used to compute a user signature, a delegate signature, or a countersignature. For a user signature or a delegate signature, the procedure returns in `row_data` the bytes in the hash in the row without any metadata.
GET_BYTES_FOR_ROW_SIGNATURE_SPECIFIED_BY_KEY_COLUMNS Procedure	It is an extension of `GET_BYTES_FOR_ROW_SIGNATURE` that uses at most three user column values instead of the `instance identifier`, `chain identifier`, and `sequence number` to uniquely identify the row.
GET_SIGNED_BLOCKCHAIN_DIGEST Function	This function generates and returns the signed digest for a specified blockchain table using the table owner's private key stored in the database wallet. The `signed_bytes`, `signed_row_indexes`, and `schema_certificate_guid` are also returned.
GET_SIGNED_BLOCKCHAIN_DIGEST_FOR_SELECTED_ROWS Function	This function generates and returns the signed digest for user-specified rows in a blockchain table using the table owner's private key stored in the database wallet. The particular rows in the digest are specified by the `row_selector` parameter. The `signed_bytes`, `signed_row_indexes`, and `schema_certificate_guid` are also returned.
SIGN_ROW Procedure	This procedure can be used by the current user to provide a signature on the row content of a previously inserted row.
SIGN_ROW_SPECIFIED_BY_KEY_COLUMNS Procedure	This procedure allows an end user or a delegate to sign a row using a set of at most three user columns that uniquely identify a single row.
SIGN_ROW_SPECIFIED_BY_KEY_COLUMNS_WITH_COUNTERSIGNATURE Procedure	This procedure uses at most three user column names and values to uniquely identify a single row to sign and countersign.

Table 43-1 (Cont.) DBMS_BLOCKCHAIN_TABLE Package Subprograms

Subprogram	Description
SIGN_ROW_WITH_COUNTERSIGNATURE Procedure	It is an extension of `SIGN_ROW` that enables the user to request a countersignature from the database. The countersignature will be produced by signing the row data content using the table owner's private key stored in the database wallet.
VERIFY_ROWS Procedure	This procedure verifies all rows on all applicable system chains for the integrity of `HASH` column value for rows created in the range of `low_timestamp` to `high_timestamp`. Row signatures can be verified as an option.
VERIFY_TABLE_BLOCKCHAIN Procedure	This procedure verifies all rows whose creation-times fall between the minimum value for the row-creation time from `bytes_previous` and the maximum value for row-creation time from `bytes_latest` and returns the number of successfully verified rows.
VERIFY_USER_BLOCKCHAIN_ROWS Procedure	This procedure verifies rows of one or more user chains when the user chains feature is enabled on the blockchain table.

ADD_INTERVAL_PARTITIONING Procedure

This procedure adds interval partitioning to an existing, non-partitioned, V1 or V2 blockchain table.

Syntax

```
DBMS_BLOCKCHAIN_TABLE.ADD_INTERVAL_PARTITIONING(
    schema_name               IN    VARCHAR2,
    table_name                IN    VARCHAR2,
    interval_number           IN    NUMBER,
    interval_frequency        IN    VARCHAR2,
    first_high_timestamp      IN    TIMESTAMP);
```

Parameters

Table 43-2 ADD_INTERVAL_PARTITIONING Parameters

Parameter	Description
`schema_name`	The name of the schema.
`table_name`	The name of the blockchain table.
`interval_number`	Sets how often the database creates partitions for the blockchain table.
`interval_frequency`	Sets the frequency for the value that was set in the `interval_number` setting. Supported values are `YEAR`, `MONTH`, `DAY`, `HOUR`, and `MINUTE`.
`first_high_timestamp`	A timestamp that determines the upper boundary of the first partition in the blockchain table.

Usage Notes

- For an existing, non-partitioned, V1 or V2 immutable table, a procedure with the same name and the same parameters is provided in the `DBMS_IMMUTABLE_TABLE` package.

- Composite partitioning (that is, sub-partitioning) is not supported with the above interval partitioning.

COUNTERSIGN_ROW Procedure

This procedure procures a countersignature on a specified row in a blockchain table. The countersignature will be produced by signing the row data content using the table owner's private key stored in the database wallet. A row in a blockchain table can be countersigned only if the row belongs to the current epoch for the blockchain table.

Syntax

```
DBMS_BLOCKCHAIN_TABLE.COUNTERSIGN_ROW (
   schema_name                      IN VARCHAR2,
   table_name                       IN VARCHAR2,
   instance_id                      IN NUMBER,
   chain_id                         IN NUMBER,
   sequence_id                      IN NUMBER,
   countersignature_algo            IN NUMBER DEFAULT
DBMS_BLOCKCHAIN_TABLE.SIGN_ALGO_DEFAULT,
   countersignature_signed_bytes    IN OUT BLOB,
   countersignature                 OUT RAW,
   countersignature_certificate_guid OUT RAW,
   countersignature_content_version  IN VARCHAR2 DEFAULT 'V2_DIGEST',
   pdb_guid                         IN RAW DEFAULT NULL);
```

Parameters

Table 43-3 COUNTERSIGN_ROW Parameters

Parameter	Description
schema_name	The name of the schema.
table_name	The name of the blockchain table.
instance_id	The instance that inserted the row. Valid values are 1, 2, and so on.
chain_id	The chain containing the row. By default, there are 32 chains in each instance, and they are numbered from 0 to 31.
sequence_id	The position of the row on the specified chain.
countersignature_algo	The digital signature algorithm for the countersignature. The parameter must be one of the following package constants: • SIGN_ALGO_RSA_SHA2_256 • SIGN_ALGO_RSA_SHA2_384 • SIGN_ALGO_RSA_SHA2_512
countersignature_signed_bytes	The bytes that are input to the algorithm that generates the countersignature. The caller must pass an empty BLOB for this parameter.
countersignature	The digital signature on the bytes returned in countersignature_signed_bytes.
countersignature_certificate_guid	A unique identifier for the certificate of the blockchain table owner stored in the database that may be used to verify the countersignature.
countersignature_content_version	The version of the data contents and layout that are used as input to the countersignature algorithm. Only 'V2_DIGEST' is supported in this release.
pdb_guid	The identifier of the local pluggable database. This parameter is used by Oracle GoldenGate replication and must be NULL.

COUNTERSIGN_ROW_SPECIFIED_BY_KEY_COLUMNS Procedure

It is an extension of `COUNTERSIGN_ROW` that uses at most three user columns to identify exactly one row in a blockchain table and procure a countersignature on that row. A row in a blockchain table can be countersigned only if the row belongs to the current epoch for the blockchain table.

Syntax

```
DBMS_BLOCKCHAIN_TABLE.COUNTERSIGN_ROW_SPECIFIED_BY_KEY_COLUMNS(
   schema_name                       IN      VARCHAR2,
   table_name                        IN      VARCHAR2,
   keycol1_name                      IN      VARCHAR2,
   keycol1_value                     IN      VARCHAR2,
   keycol2_name                      IN      VARCHAR2 DEFAULT NULL,
   keycol2_value                     IN      VARCHAR2 DEFAULT NULL,
   keycol3_name                      IN      VARCHAR2 DEFAULT NULL,
   keycol3_value                     IN      VARCHAR2 DEFAULT NULL,
   countersignature_algo             IN      NUMBER DEFAULT
DBMS_BLOCKCHAIN_TABLE.SIGN_ALGO_DEFAULT,
   countersignature_signed_bytes     IN OUT  BLOB,
   countersignature                  OUT     RAW,
   countersignature_certificate_guid OUT     RAW,
   countersignature_content_version  IN      VARCHAR2 DEFAULT 'V2_DIGEST',
   pdb_guid                          IN      RAW DEFAULT NULL);
```

Parameters

Table 43-4 COUNTERSIGN_ROW_SPECIFIED_BY_KEY_COLUMNS Procedure Parameters

Parameter	Description
`schema_name`	The name of the schema.
`table_name`	The name of the blockchain table.
`keycol1_name`	The name of the key column.
`keycol1_value`	The value of the key column.
`keycol2_name`	The name of the second column in a composite key.
`keycol2_value`	The value of the second column in a composite key.
`keycol3_name`	The name of the third column in a composite key.
`keycol3_value`	The value of the third column in a composite key.
`countersignature_algo`	The digital signature algorithm for the countersignature. The parameter must be one of the following package constants: • `SIGN_ALGO_RSA_SHA2_256` • `SIGN_ALGO_RSA_SHA2_384` • `SIGN_ALGO_RSA_SHA2_512`
`countersignature_signed_bytes`	The bytes that are input to the algorithm that generates the countersignature. The caller must pass an empty BLOB for this parameter.
`countersignature`	The digital signature on the bytes returned in `countersignature_signed_bytes`.

Table 43-4 (Cont.) COUNTERSIGN_ROW_SPECIFIED_BY_KEY_COLUMNS Procedure Parameters

Parameter	Description
countersignature_certificate_guid	A unique identifier for the certificate of the blockchain table owner stored in the database that may be used to verify the countersignature.
countersignature_content_version	The version of the data contents and layout that are used as input to the countersignature algorithm. Only 'V2_DIGEST' is supported in this release.
pdb_guid	The identifier of the local pluggable database. This parameter is used by Oracle GoldenGate replication and must be NULL.

DELETE_EXPIRED_ROWS Procedure

This procedure deletes rows outside the retention window created before_timestamp if the timestamp is specified; otherwise, this procedure deletes all rows outside the retention window. The number of rows deleted is returned in number_of_rows_deleted parameter. This procedure commits before deleting any expired rows and commits after deleting any expired rows.

Syntax

```
DBMS_BLOCKCHAIN_TABLE.DELETE_EXPIRED_ROWS(
   schema_name             IN VARCHAR2,
   table_name                 IN VARCHAR2,
   before_timestamp         IN TIMESTAMP WITH TIME ZONE DEFAULT NULL,
   number_of_rows_deleted   OUT NUMBER);
```

Parameters

Table 43-5 DELETE_EXPIRED_ROWS Procedure Parameters

Parameter	Description
schema_name	The name of the schema.
table_name	The name of the blockchain table.
before_timestamp	The end time for the range of rows deleted by the procedure, subject to the row retention time currently associated with the blockchain table. This is an optional parameter. The default value is NULL.
number_of_rows_deleted	The count of the number of rows deleted.

GET_BLOCKCHAIN_DIGEST Function

This function generates and returns a cryptographic hash of the digest for a specified blockchain table.

> **Note:**
>
> Rows inserted into a blockchain table that have not been committed are not guaranteed to be durable. Therefore, a blockchain digest does not include inserted rows that have not been committed.

Syntax

```
DBMS_BLOCKCHAIN_TABLE.GET_BLOCKCHAIN_DIGEST(
  schema_name             IN      VARCHAR2,
  table_name              IN      VARCHAR2,
  digest_bytes            IN OUT  BLOB,
  digest_rows_indexes     OUT     SYS.ORABCTAB_ROW_ARRAY_T,
  hash_algo               IN      NUMBER   DEFAULT
DBMS_BLOCKCHAIN_TABLE.HASH_ALGO_DEFAULT)
  RETURN RAW;
```

Parameters

Table 43-6 GET_BLOCKCHAIN_DIGEST Function Parameters

Parameter	Description
schema_name	The name of the schema.
table_name	The name of the blockchain table.
digest_bytes	The BLOB value that contains a header followed by an array of row-info. This sequence of bytes is input to the cryptographic hash function. The caller must pass an empty BLOB for this parameter.
digest_rows_indexes	This parameter specifies the rows in the blockchain table that were chosen for the digest.
hash_algo	The cryptographic hash algorithm to use. The parameter must be one of the following package constants: • HASH_ALGO_SHA2_256 • HASH_ALGO_SHA2_384 • HASH_ALGO_SHA2_512

Usage Notes

- A blockchain table digest created by the GET_BLOCKCHAIN_DIGEST function has table information specific to a pluggable database. Such a digest can be used only in the pluggable database in which the digest was created and only for the table that was used to create the digest. For DBMS_BLOCKCHAIN_TABLE.VERIFY_TABLE_BLOCKCHAIN, these requirements mean that both blockchain table digests must have been generated in the current pluggable database for the same blockchain table.

  For example, suppose you create a digest for a blockchain table in pluggable database A, use Data Pump to export the blockchain table, and use Data Pump to import the blockchain table into pluggable database B. The blockchain table digest created in

pluggable database A cannot be used in pluggable database B. You need to create a new blockchain table digest in pluggable database B.

> **See Also:**
>
> *Oracle Database Administrator's Guide*

GET_BLOCKCHAIN_DIGEST_FOR_SELECTED_ROWS Function

This function generates and returns a cryptographic hash of the digest for user-specified rows in a blockchain table.

> **Note:**
>
> Rows inserted into a blockchain table that have not been committed are not guaranteed to be durable. Therefore, a blockchain digest does not include inserted rows that have not been committed.

Syntax

```
DBMS_BLOCKCHAIN_TABLE.GET_BLOCKCHAIN_DIGEST_FOR_SELECTED_ROWS(
  schema_name             IN      VARCHAR2,
  table_name              IN      VARCHAR2,
  row_selector            IN      VARCHAR2,
  digest_bytes            IN OUT  BLOB,
  row_data_bytes          IN OUT  BLOB,
  digest_rows_indexes     OUT     SYS.ORABCTAB_ROW_ARRAY_T,
  hash_algo               IN      NUMBER   DEFAULT
DBMS_BLOCKCHAIN_TABLE.HASH_ALGO_DEFAULT)
  RETURN RAW;
```

Parameters

Table 43-7 GET_BLOCKCHAIN_DIGEST_FOR_SELECTED_ROWS Function Parameters

Parameter	Description
`schema_name`	The name of the schema.
`table_name`	The name of the blockchain table.
`row_selector`	The condition, a where clause without the `WHERE` keyword.
`digest_bytes`	The BLOB value that contains a header followed by an array of row-info. This sequence of bytes is input to the cryptographic hash function. The caller must pass an empty BLOB for this parameter.
`row_data_bytes`	This parameter specifies the content of the rows in the blockchain table that were selected for the digest. The caller must pass an empty BLOB for this parameter.
`digest_rows_indexes`	This parameter specifies the rows in the blockchain table that were selected for the digest.

Table 43-7 (Cont.) GET_BLOCKCHAIN_DIGEST_FOR_SELECTED_ROWS Function Parameters

Parameter	Description
hash_algo	The cryptographic hash algorithm to use. The parameter must be one of the following package constants: • HASH_ALGO_SHA2_256 • HASH_ALGO_SHA2_384 • HASH_ALGO_SHA2_512

Usage Notes

- A blockchain table digest created by the GET_BLOCKCHAIN_DIGEST_FOR_SELECTED_ROWS function has table information specific to a pluggable database. Such a digest can be used only in the pluggable database in which the digest was created and only for the table that was used to create the digest. For DBMS_BLOCKCHAIN_TABLE.VERIFY_TABLE_BLOCKCHAIN, these requirements mean that both blockchain table digests must have been generated in the current pluggable database for the same blockchain table.

  For example, suppose you create a digest for a blockchain table in pluggable database A, use Data Pump to export the blockchain table, and use Data Pump to import the blockchain table into pluggable database B. The blockchain table digest created in pluggable database A cannot be used in pluggable database B. You need to create a new blockchain table digest in pluggable database B.

See Also:

Oracle Database Administrator's Guide

GET_BYTES_FOR_ROW_HASH Procedure

This procedure returns in row_data the bytes for the particular row identified (a series of meta-data-value, column-data-value pairs in column position order) followed by the hash for the previous row in the chain in the data format specified.

Syntax

```
DBMS_BLOCKCHAIN_TABLE.GET_BYTES_FOR_ROW_HASH(
   schema_name          IN VARCHAR2,
   table_name              IN VARCHAR2,
   instance_id          IN NUMBER,
   chain_id              IN NUMBER,
   sequence_id              IN NUMBER,
   data_format              IN NUMBER,
   row_data              IN OUT BLOB,
   chain_name            IN VARCHAR2 DEFAULT NULL,
   pdb_guid              IN RAW DEFAULT NULL);
```

Parameters

Table 43-8 GET_BYTES_FOR_ROW_HASH Procedure Parameters

Parameter	Description
schema_name	The name of the schema.
table_name	The name of the blockchain table.
instance_id	The instance that inserted the row. Valid values are 1, 2, and so on.
chain_id	The chain containing the row. By default, there are 32 chains in each instance, and they are numbered from 0 to 31.
sequence_id	The position of the row on the specified chain.
data_format	The version of the data layout for the hash in the specified row. Must be 1 in this release.
row_data	The bytes for the specified row in the specified data format that can be input to the cryptographic hash function to verify the value of the hash in the row. Any bytes in the BLOB are overwritten.
chain_name	The name of the user chain when the bytes for the cryptographic hash on the user chain are desired. Specify NULL when the bytes for the cryptographic hash on the system chain are desired.
pdb_guid	For a V2 blockchain table, the identifier of the pluggable database that inserted the row. Must be NULL for a V1 blockchain table.

Usage Notes

The metadata bytes for a column are 20 bytes that encode the blockchain algorithm version used to hash the row, the column position, the column data type, whether the column value is NULL, and the actual length of the column value in bytes.

For non-character columns, the column data bytes are the actual bytes representing the column value on the disk. For character and character LOB columns, the values are normalized to specific character sets. For CHAR and NCHAR columns, the number of blanks is normalized.

A few metadata bytes are reserved for future use.

> **See Also:**
>
> For more information on normalizations, see *Oracle Database Administrator's Guide*

GET_BYTES_FOR_ROW_HASH_SPECIFIED_BY_KEY_COLUMNS Procedure

It is an extension of `GET_BYTES_FOR_ROW_HASH` that uses at most three user columns instead of an `instance identifier`, `chain identifier`, and `sequence number` to uniquely identify the row.

Syntax

```
DBMS_BLOCKCHAIN_TABLE.GET_BYTES_FOR_ROW_HASH_SPECIFIED_BY_KEY_COLUMNS(
  schema_name   IN     VARCHAR2,
  table_name    IN     VARCHAR2,
  data_format   IN     NUMBER,
  row_data      IN OUT BLOB,
  chain_name    IN     VARCHAR2 DEFAULT NULL,
  keycol1_name  IN     VARCHAR2,
  keycol1_value IN     VARCHAR2,
  keycol2_name  IN     VARCHAR2 DEFAULT NULL,
  keycol2_value IN     VARCHAR2 DEFAULT NULL,
  keycol3_name  IN     VARCHAR2 DEFAULT NULL,
  keycol3_value IN     VARCHAR2 DEFAULT NULL,
  pdb_guid      IN     RAW DEFAULT NULL);
```

Parameters

Table 43-9 GET_BYTES_FOR_ROW_HASH_SPECIFIED_BY_KEY_COLUMNS Parameters

Parameter	Description
`schema_name`	The name of the schema.
`table_name`	The name of the blockchain table.
`data_format`	The version of the data layout for the hash in the specified row. Must be 1 in this release.
`row_data`	The bytes for the specified row in the specified data format that can be input to the cryptographic hash function to verify the value of the hash in the row. Any bytes in the BLOB are overwritten.
`chain_name`	The name of the user chain when the bytes for the cryptographic hash on the user chain are desired. Specify `NULL` when the bytes for the cryptographic hash on the system chain are desired.
`keycol1_name`	The name of the key column.
`keycol1_value`	The value of the key column.
`keycol2_name`	The name of the second column in a composite key.
`keycol2_value`	The value of the second column in a composite key.
`keycol3_name`	The name of the third column in a composite key.
`keycol3_value`	The value of the third column in a composite key.
`pdb_guid`	For a V2 blockchain table, the identifier of the pluggable database that inserted the row. Must be `NULL` for a V1 blockchain table.

GET_BYTES_FOR_ROW_SIGNATURE Procedure

This procedure returns the bytes used to compute a user signature, a delegate signature, or a countersignature. For a user signature or a delegate signature, the procedure returns in `row_data` the bytes in the hash in the row without any metadata. No other columns are involved, either in the row or in the previous row. For a countersignature, the routine returns in `row_data` the bytes that are input to the digital signature algorithm that computes a countersignature on the row.

Syntax

```
DBMS_BLOCKCHAIN_TABLE.GET_BYTES_FOR_ROW_SIGNATURE(
   schema_name          IN VARCHAR2,
   table_name              IN VARCHAR2,
   instance_id          IN NUMBER,
   chain_id          IN NUMBER,
   sequence_id              IN NUMBER,
   data_format              IN NUMBER,
   row_data              IN OUT BLOB,
   pdb_guid              IN RAW DEFAULT NULL,
   signature_type        IN VARCHAR2 DEFAULT 'USER');
```

Parameters

Table 43-10 GET_BYTES_FOR_ROW_SIGNATURE Procedure Parameters

Parameter	Description
`schema_name`	The name of the schema.
`table_name`	The name of the blockchain table.
`instance_id`	The instance on which the row was inserted. Valid values are `1`, `2`, and so on.
`chain_id`	The chain on which the row was inserted. By default, there are 32 chains in each instance, and they are numbered from 0 to 31.
`sequence_id`	The position of the row on the chain.
`data_format`	The format of the data in `row_data`. Must be `1` in this release.
`row_data`	A sequence of bytes that must be signed. The caller must pass an empty BLOB for this parameter.
`pdb_guid`	For a V2 blockchain table, the identifier of the pluggable database that inserted the row. Must be `NULL` for a V1 blockchain table.
`signature_type`	The valid values for `signature_type` are `USER`, `DELEGATE`, and `COUNTERSIGNATURE`. `DELEGATE` and `USER` may be used interchangeably.

GET_BYTES_FOR_ROW_SIGNATURE_SPECIFIED_BY_KEY_COLUMNS Procedure

It is an extension of `GET_BYTES_FOR_ROW_SIGNATURE` that uses at most three user column values instead of an `instance identifier`, `chain identifier`, and `sequence number` to uniquely identify the row.

Syntax

```
DBMS_BLOCKCHAIN_TABLE.GET_BYTES_FOR_ROW_SIGNATURE_SPECIFIED_BY_KEY_COLUMNS(
  schema_name    IN     VARCHAR2,
```

```
  table_name     IN     VARCHAR2,
  data_format    IN     NUMBER,
  row_data       IN OUT BLOB,
  keycol1_name   IN     VARCHAR2,
  keycol1_value  IN     VARCHAR2,
  keycol2_name   IN     VARCHAR2 DEFAULT NULL,
  keycol2_value  IN     VARCHAR2 DEFAULT NULL,
  keycol3_name   IN     VARCHAR2 DEFAULT NULL,
  keycol3_value  IN     VARCHAR2 DEFAULT NULL,
  pdb_guid       IN     RAW DEFAULT NULL,
  signature_type IN     VARCHAR2 DEFAULT 'USER');
```

Parameters

Table 43-11 GET_BYTES_FOR_ROW_SIGNATURE_SPECIFIED_BY_KEY_COLUMNS Procedure Parameters

Parameter	Description
schema_name	The name of the schema.
table_name	The name of the blockchain table.
data_format	The format of the data in row_data. Must be 1 in this release.
row_data	A sequence of bytes that must be signed. The caller must pass an empty BLOB for this parameter.
keycol1_name	The name of the key column.
keycol1_value	The value of the key column.
keycol2_name	The name of the second column in a composite key.
keycol2_value	The value of the second column in a composite key.
keycol3_name	The name of the third column in a composite key.
keycol3_value	The value of the third column in a composite key.
pdb_guid	For a V2 blockchain table, the identifier of the pluggable database that inserted the row. Must be NULL for a V1 blockchain table.
signature_type	The valid values for signature_type are USER, DELEGATE, and COUNTERSIGNATURE. DELEGATE and USER may be used interchangeably.

GET_SIGNED_BLOCKCHAIN_DIGEST Function

This function generates and returns the signed digest for a specified blockchain table using the table owner's private key stored in the database wallet. The signed_bytes, signed_row_indexes, and schema_certificate_guid are also returned.

> **Note:**
>
> Rows inserted into a blockchain table that have not been committed are not guaranteed to be durable. Therefore, a signed blockchain digest does not include inserted rows that have not been committed.

Syntax

```
DBMS_BLOCKCHAIN_TABLE.GET_SIGNED_BLOCKCHAIN_DIGEST(
   schema_name               IN      VARCHAR2,
   table_name                   IN      VARCHAR2,
```

```
   signed_bytes               IN OUT  BLOB,
   signed_rows_indexes        OUT     SYS.ORABCTAB_ROW_ARRAY_T,
   schema_certificate_guid    OUT     RAW,
   signature_algo             IN      NUMBER DEFAULT
DBMS_BLOCKCHAIN_TABLE.SIGN_ALGO_DEFAULT)
   RETURN RAW;
```

Parameters

Table 43-12 GET_SIGNED_BLOCKCHAIN_DIGEST Function Parameters

Parameter	Description
schema_name	The name of the schema.
table_name	The name of the blockchain table.
signed_bytes	The BLOB value that contains a header followed by an array of row-info. This sequence of bytes is the digest that is digitally signed. The caller must pass an empty BLOB for this parameter.
signed_rows_indexes	This parameter specifies the rows in the blockchain table that were digitally signed.
schema_certificate_guid	A unique identifier for the certificate of the blockchain table owner stored in the database that may be used to verify the digital signature.
signature_algo	The digital signature algorithm to use. The parameter must be one of the following package constants: • SIGN_ALGO_RSA_SHA2_256 • SIGN_ALGO_RSA_SHA2_384 • SIGN_ALGO_RSA_SHA2_512

Usage Notes

- The database computes the signature on `signed_bytes` using the PKI private key of blockchain table owner.
- The certificate of blockchain table owner must be added to the database using `DBMS_USER_CERTS.ADD_CERTIFICATE`.
- The PKI private key and certificate of blockchain table owner must exist in a wallet located under the `<WALLET_ROOT>/bctable/` directory for a non-container database.
- The PKI private key and certificate of blockchain table owner must exist in a wallet located under the `<WALLET_ROOT>/pdb_guid/bctable/` directory for a container database.
- A blockchain table digest created by the `GET_SIGNED_BLOCKCHAIN_DIGEST` function has table information specific to a pluggable database. Such a digest can be used only in the pluggable database in which the digest was created and only for the table that was used to create the digest. For `DBMS_BLOCKCHAIN_TABLE.VERIFY_TABLE_BLOCKCHAIN`, these requirements mean that both blockchain table digests must have been generated in the current pluggable database for the same blockchain table.

  For example, suppose you create a digest for a blockchain table in pluggable database A, use Data Pump to export the blockchain table, and use Data Pump to import the blockchain table into pluggable database B. The blockchain table digest created in pluggable database A cannot be used in pluggable database B. You need to create a new blockchain table digest in pluggable database B.

> **Note:**
>
> The `bctable` subdirectory is the name of a database component that uses wallets. It is not the name of a blockchain table.

> **See Also:**
>
> *Oracle Database Administrator's Guide*

GET_SIGNED_BLOCKCHAIN_DIGEST_FOR_SELECTED_ROWS Function

This function generates and returns the signed digest for user-specified rows in a blockchain table using the table owner's private key stored in the database wallet. The particular rows in the digest are specified by the `row_selector` parameter. The `signed_bytes`, `signed_row_indexes`, and `schema_certificate_guid` are also returned.

> **Note:**
>
> Rows inserted into a blockchain table that have not been committed are not guaranteed to be durable. Therefore, a signed blockchain digest does not include inserted rows that have not been committed.

Syntax

```
DBMS_BLOCKCHAIN_TABLE.GET_SIGNED_BLOCKCHAIN_DIGEST_FOR_SELECTED_ROWS(
   schema_name               IN      VARCHAR2,
   table_name                  IN      VARCHAR2,
   row_selector               IN      VARCHAR2,
   signed_bytes               IN OUT  BLOB,
   row_data_bytes             IN OUT  BLOB,
   signed_rows_indexes        OUT     SYS.ORABCTAB_ROW_ARRAY_T,
   schema_certificate_guid    OUT     RAW,
   signature_algo             IN      NUMBER DEFAULT
DBMS_BLOCKCHAIN_TABLE.SIGN_ALGO_DEFAULT)
   RETURN RAW;
```

Parameters

Table 43-13 GET_SIGNED_BLOCKCHAIN_DIGEST_FOR_SELECTED_ROWS Function Parameters

Parameter	Description
`schema_name`	The name of the schema.
`table_name`	The name of the blockchain table.
`row_selector`	The condition, a where clause without the `WHERE` keyword.

Table 43-13 (Cont.) GET_SIGNED_BLOCKCHAIN_DIGEST_FOR_SELECTED_ROWS Function Parameters

Parameter	Description
signed_bytes	The BLOB value that contains a header followed by an array of row-info. This sequence of bytes is the digest that is digitally signed. The caller must pass an empty BLOB for this parameter.
row_data_bytes	This parameter specifies the content of the rows in the blockchain table that were selected for the digest. The caller must pass an empty BLOB for this parameter.
signed_rows_indexes	This parameter specifies the rows in the blockchain table that were digitally signed.
schema_certificate_guid	A unique identifier for the certificate of the blockchain table owner stored in the database that may be used to verify the digital signature.
signature_algo	The algorithm used to create the digital signature. The algorithm must be one of the following constants defined in the DBMS_BLOCKCHAIN_TABLE package: • SIGN_ALGO_RSA_SHA2_256 • SIGN_ALGO_RSA_SHA2_384 • SIGN_ALGO_RSA_SHA2_512

Usage Notes

- The database computes the signature on signed_bytes using the PKI private key of blockchain table owner.
- The certificate of blockchain table owner must be added to the database using DBMS_USER_CERTS.ADD_CERTIFICATE.
- The PKI private key and certificate of blockchain table owner must exist in a wallet located under the <WALLET_ROOT>/bctable/ directory for a non-container database.
- The PKI private key and certificate of blockchain table owner must exist in a wallet located under the <WALLET_ROOT>/pdb_guid/bctable/ directory for a container database.
- A blockchain table digest created by the GET_SIGNED_BLOCKCHAIN_DIGEST_FOR_SELECTED_ROWS function has table information specific to a pluggable database. Such a digest can be used only in the pluggable database in which the digest was created and only for the table that was used to create the digest. For DBMS_BLOCKCHAIN_TABLE.VERIFY_TABLE_BLOCKCHAIN, these requirements mean that both blockchain table digests must have been generated in the current pluggable database for the same blockchain table.

  For example, suppose you create a digest for a blockchain table in pluggable database A, use Data Pump to export the blockchain table, and use Data Pump to import the blockchain table into pluggable database B. The blockchain table digest created in pluggable database A cannot be used in pluggable database B. You need to create a new blockchain table digest in pluggable database B.

> **Note:**
>
> The bctable subdirectory is the name of a database component that uses wallets. It is not the name of a blockchain table.

> **See Also:**
>
> *Oracle Database Administrator's Guide*

SIGN_ROW Procedure

This procedure can be used by the current user to provide a signature on row content of a previously inserted row. The transaction that inserted the row into the blockchain table must have committed before the `SIGN_ROW` procedure is called.

Syntax

```
DBMS_BLOCKCHAIN_TABLE.SIGN_ROW(
   schema_name               IN VARCHAR2,
   table_name                IN VARCHAR2,
   instance_id               IN NUMBER,
   chain_id              IN NUMBER,
   sequence_id               IN NUMBER,
   hash                      IN RAW DEFAULT NULL,
   signature                 IN RAW,
   certificate_guid          IN RAW,
   signature_algo            IN NUMBER,
   delegate                  IN BOOLEAN DEFAULT FALSE,
   pdb_guid                  IN RAW DEFAULT NULL);
```

Parameters

Table 43-14 SIGN_ROW Procedure Parameters

Parameter	Description
`schema_name`	The name of the schema.
`table_name`	The name of the blockchain table.
`instance_id`	The instance on which the row was inserted.
`chain_id`	The chain containing the row to be signed. There are 32 chains in each instance, and they are numbered from 0 to 31.
`sequence_id`	The position of the row on the chain. Valid values are `1`, `2`, and so on.
`hash`	If `non-NULL`, the expected value of the hash in the row to be signed. If `NULL`, the hash in the row to be signed is not checked.
`signature`	The user's digital signature on the hash value stored in the row.
`certificate_guid`	A unique identifier for the certificate stored in the database that may be used to verify the digital signature.
`signature_algo`	The algorithm used to create the digital signature. The algorithm must be one of the following constants defined in the `DBMS_BLOCKCHAIN_TABLE` package: • `SIGN_ALGO_RSA_SHA2_256` • `SIGN_ALGO_RSA_SHA2_384` • `SIGN_ALGO_RSA_SHA2_512`
delegate	If `TRUE`, then the row is being signed by a delegate. If `FALSE`, then the row is being signed by the user that inserted the row.
pdb_guid	The identifier of the local pluggable database. This parameter is used by Oracle GoldenGate replication and must be `NULL`.

> **Note:**
>
> For information on hidden columns in blockchain tables, see Hidden Columns in Blockchain Tables

Usage Notes

- The database will verify that:
  - if delegate is `FALSE`, the current user's `obj#` matches the value in the hidden column `ORABCTAB_USER_NUMBER$` (ensures that the user owns the row)
  - if delegate is `TRUE` and the hidden column `ORABCTAB_DELEGATE_USER_NUMBER$` is not `NULL`, the current user's `obj#` matches the value in this column
  - if delegate is `TRUE`, the current user has SIGN privilege on the blockchain table
  - the hash (if provided) matches the hash column content for the row
  - the `signature` column value for the specific row identified by `'instance_id'`, `'chain_id'`, and `'sequence_id'` is `NULL`
  - if the verification succeeds, the signature value is stored for the row.
  - The `SIGN_ROW` procedure depends on information specific to a pluggable database and is applicable only to rows that were inserted in the current pluggable database by users, applications, or utilities other than Data Pump.

    For example, suppose you insert a row into a blockchain table in pluggable database A, commit the `INSERT` transaction, use Data Pump to export the blockchain table, and use Data Pump to import the blockchain table into pluggable database B. If you try to sign this row in pluggable database B, `DBMS_BLOCKCHAIN_TABLE.SIGN_ROW` will raise an exception. Hence you should sign all rows in a blockchain table that need to be signed before using Data Pump to create a copy of the blockchain table.

SIGN_ROW_SPECIFIED_BY_KEY_COLUMNS Procedure

This procedure allows an end user or a delegate to sign a row using a set of at most three user columns that uniquely identify a single row.

Syntax

```
DBMS_BLOCKCHAIN_TABLE.SIGN_ROW_SPECIFIED_BY_KEY_COLUMNS(
    schema_name      IN      VARCHAR2,
    table_name       IN      VARCHAR2,
    hash             IN      RAW DEFAULT NULL,
    signature        IN      RAW,
    certificate_guid IN      RAW,
    signature_algo   IN      NUMBER,
    delegate         IN      BOOLEAN DEFAULT FALSE,
    keycol1_name     IN      VARCHAR2,
    keycol1_value    IN      VARCHAR2,
    keycol2_name     IN      VARCHAR2 DEFAULT NULL,
    keycol2_value    IN      VARCHAR2 DEFAULT NULL,
    keycol3_name     IN      VARCHAR2 DEFAULT NULL,
    keycol3_value    IN      VARCHAR2 DEFAULT NULL,
    pdb_guid         IN      RAW DEFAULT NULL);
```

Parameters

Table 43-15 SIGN_ROW_SPECIFIED_BY_KEY_COLUMNS Procedure Parameters

Parameter	Description
schema_name	The name of the schema.
table_name	The name of the blockchain table.
hash	If non-NULL, the expected value of the hash in the row to be signed. If NULL, the hash in the row to be signed is not checked.
signature	The user's digital signature on the hash value stored in the row.
certificate_guid	A unique identifier for the certificate stored in the database that may be used to verify the digital signature.
signature_algo	The algorithm used to create the digital signature. The algorithm must be one of the following constants defined in the DBMS_BLOCKCHAIN_TABLE package: • SIGN_ALGO_RSA_SHA2_256 • SIGN_ALGO_RSA_SHA2_384 • SIGN_ALGO_RSA_SHA2_512
delegate	If TRUE, then the row is being signed by a delegate. If FALSE, then the row is being signed by the user that inserted the row.
keycol1_name	The name of the key column.
keycol1_value	The value of the key column.
keycol2_name	The name of the second column in a composite key.
keycol2_value	The value of the second column in a composite key.
keycol3_name	The name of the third column in a composite key.
keycol3_value	The value of the third column in a composite key.
pdb_guid	The identifier of the local pluggable database. This parameter is used by Oracle GoldenGate replication and must be NULL.

Usage Notes

Only non-NULL values can be used for keycol1_value, keycol2_value, and keycol3_value. If a NULL value is specified for keycol2_value or keycol3_value, then the corresponding parameter keycol2_name or keycol3_name must be NULL. Any other combination raises an exception.

SIGN_ROW_SPECIFIED_BY_KEY_COLUMNS_WITH_COUNTERSIGNATURE Procedure

This procedure uses at most three user column names and values to uniquely identify a single row to sign and countersign. A row in a blockchain table can be countersigned only if the row belongs to the current epoch for the blockchain table.

Syntax

```
DBMS_BLOCKCHAIN_TABLE.SIGN_ROW_SPECIFIED_BY_KEY_COLUMNS_WITH_COUNTERSIGNATURE (
   schema_name                    IN      VARCHAR2,
   table_name                     IN      VARCHAR2,
   hash                           IN      RAW DEFAULT NULL,
   signature                      IN      RAW,
   certificate_guid               IN      RAW,
   signature_algo                 IN      NUMBER,
```

```
   delegate                          IN      BOOLEAN DEFAULT FALSE,
   keycol1_name                      IN      VARCHAR2,
   keycol1_value                     IN      VARCHAR2,
   keycol2_name                      IN      VARCHAR2 DEFAULT NULL,
   keycol2_value                     IN      VARCHAR2 DEFAULT NULL,
   keycol3_name                      IN      VARCHAR2 DEFAULT NULL,
   keycol3_value                     IN      VARCHAR2 DEFAULT NULL,
   countersignature_algo             IN      NUMBER DEFAULT
DBMS_BLOCKCHAIN_TABLE.SIGN_ALGO_DEFAULT,
   countersignature_signed_bytes     IN      OUT BLOB,
   countersignature                  OUT     RAW,
   countersignature_certificate_guid OUT     RAW,
   countersignature_content_version  IN      VARCHAR2 DEFAULT 'V2_DIGEST',
   pdb_guid                          IN      RAW DEFAULT NULL);
```

Parameters

Table 43-16 SIGN_ROW_SPECIFIED_BY_KEY_COLUMNS_WITH_COUNTERSIGNATURE Procedure Parameters

Parameter	Description
schema_name	The name of the schema.
table_name	The name of the blockchain table.
hash	If non-NULL, the expected value of the hash in the row to be signed. If NULL, the hash in the row to be signed is not checked
signature	The user's digital signature on the hash value stored in the row.
certificate_guid	A unique identifier for the certificate stored in the database that may be used to verify the digital signature.
signature_algo	The algorithm used to create the digital signature. The algorithm must be one of the following constants defined in the DBMS_BLOCKCHAIN_TABLE package: • SIGN_ALGO_RSA_SHA2_256 • SIGN_ALGO_RSA_SHA2_384 • SIGN_ALGO_RSA_SHA2_512
delegate	If TRUE, then the row is being signed by a delegate. If FALSE, then the row is being signed by the user that inserted the row.
keycol1_name	The name of the key column.
keycol1_value	The value of the key column.
keycol2_name	The name of the second column in a composite key.
keycol2_value	The value of the second column in a composite key.
keycol3_name	The name of the third column in a composite key.
keycol3_value	The value of the third column in a composite key.
countersignature_algo	The digital signature algorithm for the countersignature. The parameter must be one of the following package constants: • SIGN_ALGO_RSA_SHA2_256 • SIGN_ALGO_RSA_SHA2_384 • SIGN_ALGO_RSA_SHA2_512
countersignature_signed_bytes	The bytes that are input to the algorithm that generates the countersignature. The caller must pass an empty BLOB for this parameter.
countersignature	The digital signature on the bytes returned in countersignature_signed_bytes.
countersignature_certificate_guid	A unique identifier for the certificate of the blockchain table owner stored in the database that may be used to verify the countersignature.

Table 43-16 (Cont.) SIGN_ROW_SPECIFIED_BY_KEY_COLUMNS_WITH_COUNTERSIGNATURE Procedure Parameters

Parameter	Description
countersignature_content_version	The version of the data contents and layout that are used as input to the countersignature algorithm. Only 'V2_DIGEST' is supported in this release.
pdb_guid	The identifier of the local pluggable database. This parameter is used by Oracle GoldenGate replication and must be NULL.

SIGN_ROW_WITH_COUNTERSIGNATURE Procedure

It is an extension of SIGN_ROW that enables the user to request a countersignature from the database. The countersignature will be produced by signing the row data content using the table owner's private key stored in the database wallet. A row in a blockchain table can be countersigned only if the row belongs to the current epoch for the blockchain table.

If delegate is TRUE, then it is the delegate user that is requesting the countersignature.

The countersignature is stored in the row along with all the meta-information needed to recompute the signed bytes used in its computation.

Syntax

```
DBMS_BLOCKCHAIN_TABLE.SIGN_ROW_WITH_COUNTERSIGNATURE(
   schema_name                       IN VARCHAR2,
   table_name                        IN VARCHAR2,
   instance_id                       IN NUMBER,
   chain_id                          IN NUMBER,
   sequence_id                       IN NUMBER,
   hash                              IN RAW DEFAULT NULL,
   signature                         IN RAW,
   certificate_guid                  IN RAW,
   signature_algo                    IN NUMBER,
   delegate                          IN BOOLEAN DEFAULT FALSE,
   countersignature_algo             IN NUMBER DEFAULT
DBMS_BLOCKCHAIN_TABLE.SIGN_ALGO_DEFAULT,
   countersignature_signed_bytes     IN OUT BLOB,
   countersignature                  OUT RAW,
   countersignature_certificate_guid OUT RAW,
   countersignature_content_version  IN VARCHAR2 DEFAULT 'V2_DIGEST',
   pdb_guid                          IN RAW DEFAULT NULL);
```

Parameters

Table 43-17 SIGN_ROW_WITH_COUNTERSIGNATURE Procedure Parameters

Parameter	Description
schema_name	The name of the schema.
table_name	The name of the blockchain table.
instance_id	The instance on which the row was inserted.

Table 43-17 (Cont.) SIGN_ROW_WITH_COUNTERSIGNATURE Procedure Parameters

Parameter	Description
chain_id	The chain containing the row to be signed. By default, there are 32 chains in each instance, and they are numbered from 0 to 31.
sequence_id	The position of the row on the chain. Valid values are 1, 2, and so on.
hash	If non-NULL, the expected value of the hash in the row to be signed. If NULL, the hash in the row to be signed is not checked.
signature	The user's digital signature on the hash value stored in the row.
certificate_guid	A unique identifier for the certificate stored in the database that may be used to verify the digital signature.
signature_algo	The algorithm used to create the digital signature. The algorithm must be one of the following constants defined in the DBMS_BLOCKCHAIN_TABLE package: • SIGN_ALGO_RSA_SHA2_256 • SIGN_ALGO_RSA_SHA2_384 • SIGN_ALGO_RSA_SHA2_512
delegate	If TRUE, then the row is being signed by a delegate. If FALSE, then the row is being signed by the user that inserted the row.
countersignature_algo	The digital signature algorithm for the countersignature. The parameter must be one of the following package constants: • SIGN_ALGO_RSA_SHA2_256 • SIGN_ALGO_RSA_SHA2_384 • SIGN_ALGO_RSA_SHA2_512
countersignature_signed_bytes	The bytes that are input to the algorithm that generates the countersignature. The caller must pass an empty BLOB for this parameter.
countersignature	The digital signature on the bytes returned in countersignature_signed_bytes.
countersignature_certificate_guid	A unique identifier for the certificate of the blockchain table owner stored in the database that may be used to verify the countersignature.
countersignature_content_version	The version of the data contents and layout that are used as input to the countersignature algorithm. Only 'V2_DIGEST' is supported in this release.
pdb_guid	The identifier of the local pluggable database. This parameter is used by Oracle GoldenGate replication and must be NULL.

Usage Notes

- SYS can always countersign any row of a blockchain table.
- An end-user or a delegate that signs and procures a countersignature using SIGN_ROW_SPECIFIED_BY_KEY_COLUMNS_WITH_COUNTERSIGNATURE or SIGN_ROW_WITH_COUNTERSIGNATURE does not need any additional privileges to procure a countersignature.

VERIFY_ROWS Procedure

Verifies all rows on all applicable system chains for integrity of `HASH` column value and optionally the `SIGNATURE` column value for rows created in the range of `low_timestamp` to `high_timestamp`. An appropriate exception is thrown if the integrity of chains is compromised.

Syntax

```
DBMS_BLOCKCHAIN_TABLE.VERIFY_ROWS(
   schema_name                  IN VARCHAR2,
   table_name                   IN VARCHAR2,
   low_timestamp                IN TIMESTAMP WITH TIME ZONE DEFAULT NULL,
   high_timestamp               IN TIMESTAMP WITH TIME ZONE DEFAULT NULL,
   instance_id                  IN NUMBER DEFAULT NULL,
   chain_id                     IN NUMBER DEFAULT NULL,
   number_of_rows_verified      OUT NUMBER,
   verify_signature             IN BOOLEAN DEFAULT TRUE,
   verify_delegate_signature    IN BOOLEAN DEFAULT TRUE,
   verify_countersignature      IN BOOLEAN DEFAULT TRUE,
   pdb_guid                     IN RAW DEFAULT NULL);
```

Parameters

Table 43-18 VERIFY_ROWS Procedure Parameters

Parameter	Description
`schema_name`	The name of the schema.
`table_name`	The name of the blockchain table.
`low_timestamp`	If specified, the low end of the time range for verifying rows. The default value is `NULL`.
`high_timestamp`	If specified, the high end of the time range for verifying rows. The default value is `NULL`.
`instance_id`	If specified, restricts row verification to rows inserted on the specified instance.
`chain_id`	If specified, restricts row verification to rows on the specified chain. By default, there are 32 chains in each instance, and they are numbered from 0 to 31.
`number_of_rows_verified`	The number of rows verified.
`verify_signature`	If `verify_signature` is `TRUE`, both the hash on each row and any user signature on the row are verified. If `verify_signature` is `FALSE`, user signatures are not verified.
`verify_delegate_signature`	If `verify_delegate_signature` is `TRUE`, both the hash on each row and any delegate signature on the row are verified. If `verify_delegate_signature` is `FALSE`, delegate signatures are not verified.
`verify_countersignature`	If `verify_countersignature` is `TRUE`, both the hash on each row and any countersignature on the row are verified. If `verify_countersignature` is `FALSE`, countersignatures are not verified.
`pdb_guid`	For a V2 blockchain table, if not `NULL`, restricts attention to system chains inserted by the specified pluggable database. Must be `NULL` for a V1 blockchain table.

Usage Notes

- `schema_name` and `table_name` are the required input parameters.

- All other input parameters are optional, with the following exception:
  - If `chain_id` is specified, `instance_id` must be specified.
- Valid values for `instance_id` are `1`, `2`, … etc.
- If neither `instance_id`, nor `chain_id` is specified, then it implies all chains. If only `instance_id` is specified, then it implies all chains on that instance. If both are specified, then it implies the specific chain provided by the combination.
- If both `low_timestamp` and `high_timestamp` are specified, then `high_timestamp` must be later than `low_timestamp`.

  If `low_timestamp` is not specified, then the range is the oldest row in the blockchain table to `high_timestamp`.

  If `high_timestamp` is not specified, then the range is `low_timestamp` to the timestamp of the last row inserted in the table.

VERIFY_TABLE_BLOCKCHAIN Procedure

This procedure verifies signatures and system chains for all rows whose creation-times fall between the minimum value for the row-creation time from `bytes_previous` and the maximum value for row-creation time from `bytes_latest`. The OUT parameter `number_of_rows_verified` returns the number of successfully verified rows.

Syntax

```
DBMS_BLOCKCHAIN_TABLE.VERIFY_TABLE_BLOCKCHAIN(
  bytes_latest              IN    BLOB      DEFAULT NULL,
  bytes_previous            IN    BLOB      DEFAULT NULL,
  number_of_rows_verified   OUT   NUMBER,
  verify_signature          IN    BOOLEAN   DEFAULT TRUE,
  verify_delegate_signature IN    BOOLEAN   DEFAULT TRUE,
  verify_countersignature   IN    BOOLEAN   DEFAULT TRUE,
  signed_bytes_latest       IN    BLOB      DEFAULT NULL,
  signed_bytes_previous     IN    BLOB      DEFAULT NULL);
```

Parameters

Table 43-19 VERIFY_TABLE_BLOCKCHAIN Procedure Parameters

Parameter	Description
`bytes_latest`	A digest populated by a call to `GET_SIGNED_BLOCKCHAIN_DIGEST`, `GET_SIGNED_BLOCKCHAIN_DIGEST_FOR_SELECTED_ROWS`, `GET_BLOCKCHAIN_DIGEST`, or `GET_BLOCKCHAIN_DIGEST_FOR_SELECTED_ROWS`.
`bytes_previous`	A digest populated by a call to `GET_SIGNED_BLOCKCHAIN_DIGEST`, `GET_SIGNED_BLOCKCHAIN_DIGEST_FOR_SELECTED_ROWS`, `GET_BLOCKCHAIN_DIGEST`, or `GET_BLOCKCHAIN_DIGEST_FOR_SELECTED_ROWS` before the `bytes_latest` BLOB was populated.
`number_of_rows_verified`	The count of the rows in the blockchain table that were verified.
`verify_signature`	If `verify_signature` is `TRUE`, both the hash on each row and any user signature on the row are verified. If `verify_signature` is `FALSE`, user signatures are not verified.

Table 43-19 (Cont.) VERIFY_TABLE_BLOCKCHAIN Procedure Parameters

Parameter	Description
verify_delegate_signature	If verify_delegate_signature is TRUE, both the hash on each row and any delegate signature on the row are verified. If verify_delegate_signature is FALSE, delegate signatures are not verified.
verify_countersignature	If verify_countersignature is TRUE, both the hash on each row and any countersignature on the row are verified. If verify_countersignature is FALSE, countersignatures are not verified.
signed_bytes_latest	signed_bytes_latest has been deprecated.
signed_bytes_previous	signed_bytes_previous has been deprecated.

Usage Notes

The BLOBs in bytes_latest and bytes_previous must be associated with the same blockchain table. For GET_SIGNED_BLOCKCHAIN_DIGEST and GET_SIGNED_BLOCKCHAIN_DIGEST_FOR_SELECTED_ROWS, the relevant BLOB parameter is called signed_bytes. For GET_BLOCKCHAIN_DIGEST and GET_BLOCKCHAIN_DIGEST_FOR_SELECTED_ROWS, the relevant BLOB parameter is called digest_bytes.

VERIFY_USER_BLOCKCHAIN_ROWS Procedure

This procedure verifies rows of one or more user chains when the user chains feature is enabled for a blockchain table.

Syntax

```
DBMS_BLOCKCHAIN_TABLE.verify_user_blockchain_rows (
   schema_name               IN    VARCHAR2,
   table_name                IN    VARCHAR2,
   row_version_name          IN    VARCHAR2,
   number_of_rows_verified   OUT   NUMBER,
   keycol1_value             IN    VARCHAR2 DEFAULT NULL,
   keycol2_value             IN    VARCHAR2 DEFAULT NULL,
   keycol3_value             IN    VARCHAR2 DEFAULT NULL,
   low_timestamp             IN    TIMESTAMP WITH TIME ZONE DEFAULT NULL,
   high_timestamp            IN    TIMESTAMP WITH TIME ZONE DEFAULT NULL,
   verify_signature          IN    BOOLEAN DEFAULT TRUE,
   verify_delegate_signature IN    BOOLEAN DEFAULT TRUE,
   verify_countersignature   IN    BOOLEAN DEFAULT TRUE,
   pdb_guid                  IN    RAW DEFAULT NULL);
```

Parameters

Table 43-20 VERIFY_USER_BLOCKCHAIN_ROWS Procedure Parameters

Parameter	Description
schema_name	The name of the schema.
table_name	The name of the blockchain table.
row_version_name	The name of the row version given when the blockchain table was created.

Table 43-20 (Cont.) VERIFY_USER_BLOCKCHAIN_ROWS Procedure Parameters

Parameter	Description
number_of_rows_verified	The number of rows verified.
keycol1_value	The value of the key column.
keycol2_value	The value of the second column in a composite key.
keycol3_value	The value of the third column in a composite key.
low_timestamp	If specified, the low end of the time range for verifying rows. The default value is NULL.
high_timestamp	If specified, the high end of the time range for verifying rows. The default value is NULL
verify_signature	If verify_signature is TRUE, both the hash on each row and any user signature on the row are verified. If verify_signature is FALSE, user signatures are not verified.
verify_delegate_signature	If verify_delegate_signature is TRUE, both the hash on each row and any delegate signature on the row are verified. If verify_delegate_signature is FALSE, delegate signatures are not verified.
verify_countersignature	If verify_countersignature is TRUE, both the hash on each row and any countersignature on the row are verified. If verify_countersignature is FALSE, countersignatures are not verified.
pdb_guid	For a V2 blockchain table, if not NULL, restricts attention to user chains inserted by the specified pluggable database. Must be NULL for a V1 blockchain table.

Usage Notes

The parameters keycol1_value, keycol2_value, and keycol3_value may be used to limit the user chains that are verified. If all three of these parameters are NULL, all user chains are verified. If one or more of these parameters are not NULL, then only user chains matching the non-NULL parameters are verified.