22

Using Oracle Flashback Technology

This chapter explains how to use Oracle Flashback Technology in database applications.

Topics:

22.1 Overview of Oracle Flashback Technology

Oracle Flashback Technology is a group of Oracle Database features that let you view past states of database objects or to return database objects to a previous state without using point-in-time media recovery.

With flashback features, you can:

- Perform queries that return past data
- Perform queries that return metadata that shows a detailed history of changes to the database
- Recover tables or rows to a previous point in time
- Automatically track and archive definitional (including schema) changes and transactional data changes
- Roll back a transaction and its dependent transactions while the database remains online

Oracle Flashback features use the Automatic Undo Management (AUM) system to obtain metadata and historical data for transactions. They rely on **undo data**, which are records of the effects of individual transactions. For example, if a user runs an `UPDATE` statement to change a salary from 1000 to 1100, then Oracle Database stores the value 1000 in the undo data.

Undo data is persistent and survives a database shutdown. It is retained for the time specified by the `undo_retention` parameter, or up to the tuned undo retention in the presence of Automatic Undo Management (AUM). By using flashback features, you can use undo data to

query past data or recover from logical damage. Besides using it in flashback features, Oracle Database uses undo data to perform these actions:

- Roll back active transactions
- Recover terminated transactions by using database or process recovery
- Provide read consistency for SQL queries

> **Note:**
>
> After executing a `CREATE TABLE` statement, wait at least 15 seconds to commit any transactions, to ensure that Oracle Flashback features (especially Oracle Flashback Version Query) reflect those transactions.

> **Note:**
>
> Oracle Database recommends to avoid the usage of `versions_starttime`, `versions_endtime` or `scn_to_timestamp` columns in
>
> `VERSIONS` queries (including `CTAS` queries) to improve the performance.

Topics:

- Application Development Features
- Database Administration Features

> **See Also:**
>
> *Oracle Database Concepts* for more information about flashback features

22.1.1 Application Development Features

In application development, you can use these flashback features to report historical data or undo erroneous changes. (You can also use these features interactively as a database user or administrator.)

Oracle Flashback Query

Use this feature to retrieve data for an earlier time that you specify with the `AS OF` clause of the `SELECT` statement.

> **See Also:**
>
> Using Oracle Flashback Query (SELECT AS OF)

Oracle Flashback Version Query

Use this feature to retrieve metadata and historical data for a specific time interval (for example, to view all the rows of a table that ever existed during a given time interval). Metadata for each row version includes start and end time, type of change operation, and identity of the transaction that created the row version. To create an Oracle Flashback Version Query, use the `VERSIONS BETWEEN` clause of the `SELECT` statement.

> **See Also:**
>
> Using Oracle Flashback Version Query

Oracle Flashback Transaction Query

Use this feature to retrieve metadata and historical data for a given transaction or for all transactions in a given time interval. To perform an Oracle Flashback Transaction Query, select from the static data dictionary view `FLASHBACK_TRANSACTION_QUERY`.

> **See Also:**
>
> Using Oracle Flashback Transaction Query.

Typically, you use Oracle Flashback Transaction Query with an Oracle Flashback Version Query that provides the transaction IDs for the rows of interest.

> **See Also:**
>
> Using Oracle Flashback Transaction Query with Oracle Flashback Version Query

DBMS_FLASHBACK Package

Use this feature to set the internal Oracle Database clock to an earlier time so that you can examine data that was current at that time, or to roll back a transaction and its dependent transactions while the database remains online.

> **See Also:**
>
> - Flashback Transaction
> - Using DBMS_FLASHBACK Package

Flashback Transaction

Use Flashback Transaction to roll back a transaction and its dependent transactions while the database remains online. This recovery operation uses undo data to create and run the

corresponding compensating transactions that return the affected data to its original state. (Flashback Transaction is part of `DBMS_FLASHBACK` package).

> **See Also:**
>
> Using DBMS_FLASHBACK Package.

Flashback Time Travel

Use Flashback Time Travel to automatically track and archive historical versions of changes to tables enabled for flashback archive, ensuring SQL-level access to the versions of database objects without getting a snapshot-too-old error.

> **See Also:**
>
> Using Flashback Time Travel.

22.1.2 Database Administration Features

These flashback features are primarily for data recovery. Typically, you use these features only as a database administrator.

This chapter focuses on the Application Development Features.

> **See Also:**
>
> - *Oracle Database Administrator's Guide*
> - *Oracle Database Backup and Recovery User's Guide*

Oracle Flashback Table

Use this feature to restore a table to its state at a previous point in time. You can restore a table while the database is on line, undoing changes to only the specified table.

Oracle Flashback Drop

Use this feature to recover a dropped table. This feature reverses the effects of a `DROP TABLE` statement.

Oracle Flashback Database

Use this feature to quickly return the database to an earlier point in time, by using the recovery area. This is fast, because you do not have to restore database backups.

22.2 Configuring Your Database for Oracle Flashback Technology

Before you can use flashback features in your application, you or your database administrator must perform the configuration tasks described in these topics:

Topics:

22.2.1 Configuring Your Database for Automatic Undo Management

To configure your database for Automatic Undo Management (AUM), you or your database administrator must:

- Create an undo tablespace with enough space to keep the required data for flashback operations.

  The more often users update the data, the more space is required. The database administrator usually calculates the space requirement.

- Enable AUM, as explained in *Oracle Database Administrator's Guide*. Set these database initialization parameters:

  - `UNDO_MANAGEMENT`
  - `UNDO_TABLESPACE`

Note:

- Additional configuration of the `UNDO_RETENTION` parameter is required only if you use Oracle Flashback operations or Active Data Guard.

  See Also:

  Setting the Minimum Undo Retention Period in *Oracle Database Administrator's Guide* for more information about the `UNDO_RETENTION` parameter

- Starting with Oracle Database release 19c, version 19.9, the value of `UNDO_RETENTION` is not inherited in a CDB.

  See Also:

  UNDO_RETENTION in *Oracle Database Reference* for more information about the `UNDO_RETENTION` parameter

For a fixed-size undo tablespace, Oracle Database automatically tunes the system to give the undo tablespace the best possible undo retention.

For an automatically extensible undo tablespace, Oracle Database retains undo data longer than the longest query duration and the low threshold of undo retention specified by the `UNDO_RETENTION` parameter.

Note:

You can query `V$UNDOSTAT.TUNED_UNDORETENTION` to determine the amount of time for which undo is retained for the current undo tablespace.

Setting `UNDO_RETENTION` does not guarantee that unexpired undo data is not discarded. If the system needs more space, Oracle Database can overwrite unexpired undo with more recently generated undo data.

- Specify the `RETENTION GUARANTEE` clause for the undo tablespace to ensure that unexpired undo data is not discarded.

See Also:

- *Oracle Database Administrator's Guide* for more information about creating an undo tablespace and enabling AUM
- *Oracle Database Reference* for more information about `V$UNDOSTAT`

22.2.2 Configuring Your Database for Oracle Flashback Transaction Query

To configure your database for the Oracle Flashback Transaction Query feature, you or your database administrator must:

- Ensure that Oracle Database is running with version 10.0 compatibility.
- Enable supplemental logging:

```
ALTER DATABASE ADD SUPPLEMENTAL LOG DATA;
```

22.2.3 Configuring Your Database for Flashback Transaction

To configure your database for the Flashback Transaction feature, you or your database administrator must:

- With the database mounted but not open, enable `ARCHIVELOG`:

```
ALTER DATABASE ARCHIVELOG;
```

- Open at least one archive log:

```
ALTER SYSTEM ARCHIVE LOG CURRENT;
```

- If not done, enable minimal and primary key supplemental logging:

```
ALTER DATABASE ADD SUPPLEMENTAL LOG DATA;
ALTER DATABASE ADD SUPPLEMENTAL LOG DATA (PRIMARY KEY) COLUMNS;
```

- If you want to track foreign key dependencies, enable foreign key supplemental logging:

```
ALTER DATABASE ADD SUPPLEMENTAL LOG DATA (FOREIGN KEY) COLUMNS;
```

> **Note:**
>
> If you have very many foreign key constraints, enabling foreign key supplemental logging might not be worth the performance penalty.

22.2.4 Enabling Oracle Flashback Operations on Specific LOB Columns

To enable flashback operations on specific LOB columns of a table, use the `ALTER TABLE` statement with the `RETENTION` option.

Because undo data for LOB columns can be voluminous, you must define which LOB columns to use with flashback operations.

> **See Also:**
>
> *Oracle Database SecureFiles and Large Objects Developer's Guide* to learn about LOB storage and the `RETENTION` parameter

22.2.5 Granting Necessary Privileges

You or your database administrator must grant privileges to users, roles, or applications that must use these flashback features.

> **See Also:**
>
> *Oracle Database SQL Language Reference* for information about the `GRANT` statement

For Oracle Flashback Query and Oracle Flashback Version Query

To allow access to specific objects during queries, grant `FLASHBACK` and either `READ` or `SELECT` privileges on those objects.

To allow queries on all tables, grant the `FLASHBACK ANY TABLE` privilege.

For Oracle Flashback Transaction Query

Grant the `SELECT ANY TRANSACTION` privilege.

To allow execution of undo SQL code retrieved by an Oracle Flashback Transaction Query, grant `SELECT`, `UPDATE`, `DELETE`, and `INSERT` privileges for specific tables.

For DBMS_FLASHBACK Package

To allow access to the features in the `DBMS_FLASHBACK` package, grant the `EXECUTE` privilege on `DBMS_FLASHBACK`.

For Flashback Time Travel

To allow a specific user to enable Flashback Time Travel on tables, using a specific Flashback Archive, grant the `FLASHBACK ARCHIVE` object privilege on that Flashback Archive to that user. To grant the `FLASHBACK ARCHIVE` object privilege, you must either be logged on as `SYSDBA` or have `FLASHBACK ARCHIVE ADMINISTER` system privilege.

To allow execution of these statements, grant the `FLASHBACK ARCHIVE ADMINISTER` system privilege:

- `CREATE FLASHBACK ARCHIVE`
- `ALTER FLASHBACK ARCHIVE`
- `DROP FLASHBACK ARCHIVE`

To grant the `FLASHBACK ARCHIVE ADMINISTER` system privilege, you must be logged on as `SYSDBA`.

To create a default Flashback Archive, using either the `CREATE FLASHBACK ARCHIVE` or `ALTER FLASHBACK ARCHIVE` statement, you must be logged on as `SYSDBA`.

To disable Flashback Archive for a table that has been enabled for Flashback Archive, you must either be logged on as `SYSDBA` or have the `FLASHBACK ARCHIVE ADMINISTER` system privilege.

22.3 Using Oracle Flashback Query (SELECT AS OF)

To use Oracle Flashback Query, use a `SELECT` statement with an `AS OF` clause. Oracle Flashback Query retrieves data as it existed at an earlier time. The query explicitly references a past time through a time stamp or System Change Number (SCN). It returns committed data that was current at that point in time.

Uses of Oracle Flashback Query include:

- Recovering lost data or undoing incorrect, committed changes.

  For example, if you mistakenly delete or update rows, and then commit them, you can immediately undo the mistake.
- Comparing current data with the corresponding data at an earlier time.

  For example, you can run a daily report that shows the change in data from yesterday. You can compare individual rows of table data or find intersections or unions of sets of rows.
- Checking the state of definitional (DDL) data or schema at a particular time.

  For example, you can issue the flashback query on a table and its schema to view the history of DDL changes made to the table.
- Checking the state of transactional data at a particular time.

  For example, you can verify the account balance of a certain day.
- Selecting data that was valid at a particular time or at any time within a user-defined valid time period.

  For example, you can find employees with valid employee information as of a particular timestamp or between a specified start and end time in the specified valid time period. (For more information, see Temporal Validity Support.)
- Simplifying application design by removing the need to store some kinds of temporal data.

  Oracle Flashback Query lets you retrieve past data directly from the database.
- Applying packaged applications, such as report generation tools, to past data.
- Providing self-service error correction for an application, thereby enabling users to undo and correct their errors.

Topics:

- Example: Examining and Restoring Past Data
- Guidelines for Oracle Flashback Query

> **See Also:**
>
> *Oracle Database SQL Language Reference* for more information about the `SELECT AS OF` statement

22.3.1 Example: Examining and Restoring Past Data

Suppose that you discover at 12:30 PM that the row for employee Chung was deleted from the `employees` table, and you know that at 9:30AM the data for Chung was correctly stored in the

database. You can use Oracle Flashback Query to examine the contents of the table at 9:30 AM to find out what data was lost. If appropriate, you can restore the lost data.

Example 22-1 retrieves the state of the record for `Chung` at 9:30AM, April 4, 2004:

Example 22-1 Retrieving a Lost Row with Oracle Flashback Query

```
SELECT * FROM employees
AS OF TIMESTAMP
TO_TIMESTAMP('2004-04-04 09:30:00', 'YYYY-MM-DD HH:MI:SS')
WHERE last_name = 'Chung';
```

Example 22-2 restores Chung's information to the `employees` table:

Example 22-2 Restoring a Lost Row After Oracle Flashback Query

```
INSERT INTO employees (
  SELECT * FROM employees
  AS OF TIMESTAMP
  TO_TIMESTAMP('2004-04-04 09:30:00', 'YYYY-MM-DD HH:MI:SS')
  WHERE last_name = 'Chung'
);
```

22.3.2 Guidelines for Oracle Flashback Query

- You can specify or omit the `AS OF` clause for each table and specify different times for different tables.

  > **Note:**
  >
  > If a table is a Flashback Time Travel and you specify a time for it that is earlier than its creation time, the query returns zero rows for that table, rather than causing an error.

- You can use the `AS OF` clause in queries to perform data definition language (DDL) operations (such as creating and truncating tables) or data manipulation language (DML) statements (such as `INSERT` and `DELETE`) in the same session as Oracle Flashback Query.
- To use the result of Oracle Flashback Query in a DDL or DML statement that affects the current state of the database, use an `AS OF` clause inside an `INSERT` or `CREATE TABLE AS SELECT` statement.
- If a possible 3-second error (maximum) is important to Oracle Flashback Query in your application, use an SCN instead of a time stamp. See General Guidelines for Oracle Flashback Technology.
- You can create a view that refers to past data by using the `AS OF` clause in the `SELECT` statement that defines the view.

  If you specify a relative time by subtracting from the current time on the database host, the past time is recalculated for each query. For example:

  ```
  CREATE VIEW hour_ago AS
    SELECT * FROM employees
      AS OF TIMESTAMP (SYSTIMESTAMP - INTERVAL '60' MINUTE);
  ```

  `SYSTIMESTAMP` returns the current date, including fractional seconds.

`SYSTIMESTAMP` uses the time zone of either the database host system or the database, depending on the setting of the `TIME_AT_DBTIMEZONE` initialization parameter. See Oracle Database Reference: TIME_AT_DBTIMEZONE for more information.

- You can use the `AS OF` clause in self-joins, or in set operations such as `INTERSECT` and `MINUS`, to extract or compare data from two different times.

  You can store the results by preceding Oracle Flashback Query with a `CREATE TABLE AS SELECT` or `INSERT INTO TABLE SELECT` statement. For example, this query reinserts into table `employees` the rows that existed an hour ago:

  ```
  INSERT INTO employees
      (SELECT * FROM employees
       AS OF TIMESTAMP (SYSTIMESTAMP - INTERVAL '60' MINUTE)
      MINUS SELECT * FROM employees;
  ```

  `SYSTIMESTAMP` returns the current date, including fractional seconds.

  `SYSTIMESTAMP` uses the time zone of either the database host system or the database, depending on the setting of the `TIME_AT_DBTIMEZONE` initialization parameter. See Oracle Database Reference: TIME_AT_DBTIMEZONE for more information.

- You can use the `AS OF` clause in queries to check for data that was valid at a particular time.

> **See Also:**
>
> - Temporal Validity Support
> - Using Flashback Time Travel for information about Flashback Time Travel

22.4 Using Oracle Flashback Version Query

Use Oracle Flashback Version Query to retrieve the different versions of specific rows that existed during a given time interval. A row version is created whenever a `COMMIT` statement is executed.

> **Note:**
>
> After executing a `CREATE TABLE` statement, wait at least 15 seconds to commit any transactions, to ensure that Oracle Flashback Version Query reflects those transactions.

Specify Oracle Flashback Version Query using the `VERSIONS BETWEEN` clause of the `SELECT` statement. The syntax is either:

```
VERSIONS BETWEEN { SCN | TIMESTAMP } start AND end
```

where *`start`* and *`end`* are expressions representing the start and end, respectively, of the time interval to be queried. The time interval includes (*`start`* and *`end`*).

or:

```
VERSIONS PERIOD FOR user_valid_time [ BETWEEN TIMESTAMP start AND end ]
```

where *user_valid_time* refers to the user-specified valid time period, as explained in Temporal Validity Support.

Oracle Flashback Version Query returns a table with a row for each version of the row that existed at any time during the specified time interval. Each row in the table includes pseudocolumns of metadata about the row version, which can reveal when and how a particular change (perhaps erroneous) occurred to your database.

Table 22-1 describes the pseudocolumns of metadata about the row version. The VERSIONS_* pseudocolumns have values only for transaction-time Flashback Version Queries (that is, queries with the clause BETWEEN TIMESTAMP *start* AND *end*).

Table 22-1 Oracle Flashback Version Query Row Data Pseudocolumns

Pseudocolumn Name	Description
VERSIONS_STARTSCN VERSIONS_STARTTIME	Starting System Change Number (SCN) or TIMESTAMP when the row version was created. This pseudocolumn identifies the time when the data first had the values reflected in the row version. Use this pseudocolumn to identify the past target time for Oracle Flashback Table or Oracle Flashback Query. If this pseudocolumn is NULL, then the row version was created before *start*.
VERSIONS_ENDSCN	SCN or TIMESTAMP when the row version expired.
VERSIONS_ENDTIME	If this pseudocolumn is NULL, then either the row version was current at the time of the query or the row corresponds to a DELETE operation.
VERSIONS_XID	Identifier of the transaction that created the row version.
VERSIONS_OPERATION	Operation performed by the transaction: I for insertion, D for deletion, or U for update. The version is that of the row that was inserted, deleted, or updated; that is, the row after an INSERT operation, the row before a DELETE operation, or the row affected by an UPDATE operation. For user updates of an index key, Oracle Flashback Version Query might treat an UPDATE operation as two operations, DELETE plus INSERT, represented as two version rows with a D followed by an I VERSIONS_OPERATION.

A given row version is valid starting at its time VERSIONS_START* up to, but not including, its time VERSIONS_END*. That is, it is valid for any time *t* such that VERSIONS_START* **<=** *t* **<** VERSIONS_END*. For example, this output indicates that the salary was 10243 from September 9, 2002, included, to November 25, 2003, excluded.

```
VERSIONS_START_TIME     VERSIONS_END_TIME     SALARY
-------------------     -----------------     ------
09-SEP-2003             25-NOV-2003           10243
```

Here is a typical use of Oracle Flashback Version Query:

```
SELECT versions_startscn, versions_starttime,
       versions_endscn, versions_endtime,
       versions_xid, versions_operation,
       last_name, salary
  FROM employees
  VERSIONS BETWEEN TIMESTAMP
      TO_TIMESTAMP('2008-12-18 14:00:00', 'YYYY-MM-DD HH24:MI:SS')
  AND TO_TIMESTAMP('2008-12-18 17:00:00', 'YYYY-MM-DD HH24:MI:SS')
  WHERE first_name = 'John';
```

You can use `VERSIONS_XID` with Oracle Flashback Transaction Query to locate this transaction's metadata, including the SQL required to undo the row change and the user responsible for the change.

Flashback Version Query allows index-only access only with IOTs (index-organized tables), but index fast full scan is not allowed.

> **See Also:**
>
> - *Oracle Database SQL Language Reference* for information about Oracle Flashback Version Query pseudocolumns and the syntax of the `VERSIONS` clause
> - Using Oracle Flashback Transaction Query

22.5 Using Oracle Flashback Transaction Query

Use Oracle Flashback Transaction Query to retrieve metadata and historical data for a given transaction or for all transactions in a given time interval. Oracle Flashback Transaction Query queries the static data dictionary view `FLASHBACK_TRANSACTION_QUERY`, whose columns are described in *Oracle Database Reference*.

The column `UNDO_SQL` shows the SQL code that is the logical opposite of the DML operation performed by the transaction. You can usually use this code to reverse the logical steps taken during the transaction. However, there are cases where the `UNDO_SQL` code is not the exact opposite of the original transaction. For example, a `UNDO_SQL INSERT` operation might not insert a row back in a table at the same `ROWID` from which it was deleted.

This statement queries the `FLASHBACK_TRANSACTION_QUERY` view for transaction information, including the transaction ID, the operation, the operation start and end SCNs, the user responsible for the operation, and the SQL code that shows the logical opposite of the operation:

```
SELECT xid, operation, start_scn, commit_scn, logon_user, undo_sql
FROM flashback_transaction_query
WHERE xid = HEXTORAW('000200030000002D');
```

This statement uses Oracle Flashback Version Query as a subquery to associate each row version with the `LOGON_USER` responsible for the row data change:

```
SELECT xid, logon_user
FROM flashback_transaction_query
WHERE xid IN (
  SELECT versions_xid FROM employees VERSIONS BETWEEN TIMESTAMP
  TO_TIMESTAMP('2003-07-18 14:00:00', 'YYYY-MM-DD HH24:MI:SS') AND
  TO_TIMESTAMP('2003-07-18 17:00:00', 'YYYY-MM-DD HH24:MI:SS')
);
```

> **Note:**
>
> If you query `FLASHBACK_TRANSACTION_QUERY` without specifying `XID` in the `WHERE` clause, the query scans many unrelated rows, degrading performance.

See Also:

- *Oracle Database Backup and Recovery User's Guide.* for information about how a database administrator can use Flashback Table to restore an entire table, rather than individual rows
- *Oracle Database Administrator's Guide* for information about how a database administrator can use Flashback Table to restore an entire table, rather than individual rows

22.6 Using Oracle Flashback Transaction Query with Oracle Flashback Version Query

In this example, a database administrator does this:

```
DROP TABLE emp;
CREATE TABLE emp (
  empno   NUMBER PRIMARY KEY,
  empname VARCHAR2(16),
  salary  NUMBER
);
INSERT INTO emp (empno, empname, salary) VALUES (111, 'Mike', 555);
COMMIT;

DROP TABLE dept;
CREATE TABLE dept (
  deptno   NUMBER,
  deptname VARCHAR2(32)
);
INSERT INTO dept (deptno, deptname) VALUES (10, 'Accounting');
COMMIT;
```

Now `emp` and `dept` have one row each. In terms of row versions, each table has one version of one row. Suppose that an erroneous transaction deletes `empno 111` from table `emp`:

```
UPDATE emp SET salary = salary + 100 WHERE empno = 111;
INSERT INTO dept (deptno, deptname) VALUES (20, 'Finance');
DELETE FROM emp WHERE empno = 111;
COMMIT;
```

Next, a transaction reinserts `empno 111` into the `emp` table with a new employee name:

```
INSERT INTO emp (empno, empname, salary) VALUES (111, 'Tom', 777);
UPDATE emp SET salary = salary + 100 WHERE empno = 111;
UPDATE emp SET salary = salary + 50 WHERE empno = 111;
COMMIT;
```

The database administrator detects the application error and must diagnose the problem. The database administrator issues this query to retrieve versions of the rows in the `emp` table that correspond to empno `111`. The query uses Oracle Flashback Version Query pseudocolumns:

```
SELECT versions_xid XID, versions_startscn START_SCN,
  versions_endscn END_SCN, versions_operation OPERATION,
  empname, salary
FROM emp
```

```
VERSIONS BETWEEN SCN MINVALUE AND MAXVALUE
WHERE empno = 111;
```

Results are similar to:

```
XID               START_SCN    END_SCN O EMPNAME              SALARY
---------------- ---------- ---------- - ---------------- ----------
09001100B2200000   10093466            I Tom                     927
030002002B210000   10093459            D Mike                    555
0800120096200000   10093375   10093459 I Mike                    555

3 rows selected.
```

The results table rows are in descending chronological order. The third row corresponds to the version of the row in the table `emp` that was inserted in the table when the table was created. The second row corresponds to the row in `emp` that the erroneous transaction deleted. The first row corresponds to the version of the row in `emp` that was reinserted with a new employee name.

The database administrator identifies transaction `030002002B210000` as the erroneous transaction and uses Oracle Flashback Transaction Query to audit all changes made by this transaction:

```
SELECT  xid, start_scn, commit_scn, operation, logon_user, undo_sql
FROM flashback_transaction_query
WHERE xid = HEXTORAW('000200030000002D');
```

Results are similar to:

```
XID               START_SCN COMMIT_SCN OPERATION LOGON_USER
---------------- ---------- ---------- --------- ------------------------------
UNDO_SQL
--------------------------------------------------------------------------------

030002002B210000   10093452   10093459 DELETE    HR
insert into "HR"."EMP"("EMPNO","EMPNAME","SALARY") values ('111','Mike','655');

030002002B210000   10093452   10093459 INSERT    HR
delete from "HR"."DEPT" where ROWID = 'AAATjuAAEAAAAJrAAB';

030002002B210000   10093452   10093459 UPDATE    HR
update "HR"."EMP" set "SALARY" = '555' where ROWID = 'AAATjsAAEAAAAJ7AAA';

030002002B210000   10093452   10093459 BEGIN     HR


4 rows selected.
```

To make the result of the next query easier to read, the database administrator uses these SQL*Plus commands:

```
COLUMN operation FORMAT A9
COLUMN table_name FORMAT A10
COLUMN table_owner FORMAT A11
```

To see the details of the erroneous transaction and all subsequent transactions, the database administrator performs this query:

```
SELECT xid, start_scn, commit_scn, operation, table_name, table_owner
FROM flashback_transaction_query
WHERE table_owner = 'HR'
```

```
AND start_timestamp >=
  TO_TIMESTAMP ('2002-04-16 11:00:00','YYYY-MM-DD HH:MI:SS');
```

Results are similar to:

```
XID              START_SCN COMMIT_SCN OPERATION TABLE_NAME TABLE_OWNER
---------------- ---------- ---------- --------- ---------- -----------
02000E0074200000   10093435   10093446 INSERT    DEPT       HR
030002002B210000   10093452   10093459 DELETE    EMP        HR
030002002B210000   10093452   10093459 INSERT    DEPT       HR
030002002B210000   10093452   10093459 UPDATE    EMP        HR
0800120096200000   10093374   10093375 INSERT    EMP        HR
09001100B2200000   10093462   10093466 UPDATE    EMP        HR
09001100B2200000   10093462   10093466 UPDATE    EMP        HR
09001100B2200000   10093462   10093466 INSERT    EMP        HR

8 rows selected.
```

> **Note:**
>
> Because the preceding query does not specify the `XID` in the `WHERE` clause, it scans many unrelated rows, degrading performance.

22.7 Using DBMS_FLASHBACK Package

The `DBMS_FLASHBACK` package provides the same functionality as Oracle Flashback Query, but Oracle Flashback Query is sometimes more convenient.

The `DBMS_FLASHBACK` package acts as a time machine: you can turn back the clock, perform normal queries as if you were at that earlier time, and then return to the present. Because you can use the `DBMS_FLASHBACK` package to perform queries on past data without special clauses such as `AS OF` or `VERSIONS BETWEEN`, you can reuse existing PL/SQL code to query the database at earlier times.

You must have the `EXECUTE` privilege on the `DBMS_FLASHBACK` package.

To use the `DBMS_FLASHBACK` package in your PL/SQL code:

1. Specify a past time by invoking either `DBMS_FLASHBACK.ENABLE_AT_TIME` or `DBMS_FLASHBACK.ENABLE_AT_SYSTEM_CHANGE_NUMBER`.
2. Perform regular queries (that is, queries without special flashback-feature syntax such as `AS OF`). Do not perform DDL or DML operations.

   The database is queried at the specified past time.
3. Return to the present by invoking `DBMS_FLASHBACK.DISABLE`.

   You must invoke `DBMS_FLASHBACK.DISABLE` before invoking `DBMS_FLASHBACK.ENABLE_AT_TIME` or `DBMS_FLASHBACK.ENABLE_AT_SYSTEM_CHANGE_NUMBER` again. You cannot nest enable/disable pairs.

To use a cursor to store the results of queries, open the cursor before invoking `DBMS_FLASHBACK.DISABLE`. After storing the results and invoking `DBMS_FLASHBACK.DISABLE`, you can:

- Perform `INSERT` or `UPDATE` operations to modify the current database state by using the stored results from the past.

- Compare current data with the past data. After invoking `DBMS_FLASHBACK.DISABLE`, open a second cursor. Fetch from the first cursor to retrieve past data; fetch from the second cursor to retrieve current data. You can store the past data in a temporary table and then use set operators such as `MINUS` or `UNION` to contrast or combine the past and current data.

You can invoke `DBMS_FLASHBACK.GET_SYSTEM_CHANGE_NUMBER` at any time to get the current System Change Number (SCN). `DBMS_FLASHBACK.GET_SYSTEM_CHANGE_NUMBER` returns the current SCN regardless of previous invocations of `DBMS_FLASHBACK.ENABLE`.

See Also:

- *Oracle Database PL/SQL Packages and Types Reference* for details of the `DBMS_FLASHBACK` package

22.7.1 Using Flashback Version Query with DBMS_FLASHBACK

You can enable `DBMS_FLASHBACK` for a session using the `DBMS_FLASHBACK.ENABLE_AT_SYSTEM_CHANGE_NUMBER` or `DBMS_FLASHBACK.ENABLE_AT_TIME` procedure. The `ENABLE_AT_SYSTEM_CHANGE_NUMBER` and `ENABLE_AT_TIME` procedures are based on System Change Number (SCN). You can use Oracle Flashback Version Query (version query) to retrieve committed row versions pertaining to a time interval. To use a version query (with the `VERSIONS BETWEEN` clause), you need to specify the lower and upper `SCN` limits, either using specific `SCN` values or using the `MINVALUE` and `MAXVALUE` keywords.

Using a version query along with `DBMS_FLASHBACK` affects the `MAXVALUE` keyword or the upper `SCN` limit in the version query, in that if the version query's upper `SCN` limit (through the `MAXVALUE` keyword or provided value) exceeds the `DBMS_FLASHBACK SCN`, the version query takes the `DBMS_FLASHBACK SCN` as the maximum limit. For example, suppose that the session-level Flashback is enabled using the `ENABLE_AT_SYSTEM_CHANGE_NUMBER` procedure with the `SCN` parameter set as `SCN_X`. A version query using `VERSIONS BETWEEN` clause is passed later with the minimum and maximum limits as `SCN_A` and `SCN_B`. When the version query runs, the `SCN`s are treated in the following manner:

- When the versions query specifies a `MAXVALUE` clause (instead of a specific `SCN_B`), `SCN_X` is treated as the maximum limit.
- When the versions query has two `SCN`s, `SCN_A` and `SCN_B`, and `SCN_B` is less than `SCN_X`, then the `SCN`s in the versions query are retained without any changes.
- When the versions query has two `SCN`s, `SCN_C` and `SCN_D`,and `SCN_D` is greater than `SCN_X`, then `SCN_X` is treated as the maximum limit.

The following code sample shows how a version query could be used after `DBMS_FLASHBACK` is enabled for a session.

```
DBMS_FLASHBACK.ENABLE_AT_SYSTEM_CHANGE_NUMBER(123);
…
…
SELECT * FROM <table_name> VERSIONS BETWEEN SCN 120 and 125;
…
DBMS_FLASHBACK.DISABLE();
```

Here, the upper `SCN` limit of 125 in the version query is replaced with `DBMS_FLASHABCK SCN` of 123 because the version query's upper `SCN` limit is greater than the `DBMS_FLASHABCK SCN`. Therefore, the following is the resultant version query:

```
SELECT * FROM <table_name> VERSIONS BETWEEN SCN 120 and 123;
```

> **See Also:**
>
> - *Oracle Database PL/SQL Packages and Types Reference* for information about the `DBMS_FLASHBACK` package
> - *Oracle Database SQL Language Reference* for information about Oracle Flashback Version Query

22.8 Using Flashback Transaction

The `DBMS_FLASHBACK.TRANSACTION_BACKOUT` procedure rolls back a transaction and its dependent transactions while the database remains online. This recovery operation uses undo data to create and run the **compensating transactions** that return the affected data to its original state.

The transactions being rolled back are subject to these restrictions:

- They cannot have performed DDL operations that changed the logical structure of database tables.
- They cannot use Large Object (LOB) Data Types:
  - `BFILE`
  - `BLOB`
  - `CLOB`
  - `NCLOB`
- They cannot use features that LogMiner does not support.

  The features that LogMiner supports depends on the value of the `COMPATIBLE` initialization parameter for the database that is rolling back the transaction. The default value is the release number of the most recent major release.

  Flashback Transaction inherits SQL data type support from LogMiner. Therefore, if LogMiner fails due to an unsupported SQL data type in a the transaction, Flashback Transaction fails too.

  Some data types, though supported by LogMiner, do not generate undo information as part of operations that modify columns of such types. Therefore, Flashback Transaction does not support tables containing these data types. These include tables with BLOB, CLOB and XML type.

> **See Also:**
>
> - *Oracle Data Guard Concepts and Administration* for information about data type and DDL support on a logical standby database
> - *Oracle Database SQL Language Reference* for information about LOB data types
> - *Oracle Database Utilities* for information about LogMiner
> - *Oracle Database Administrator's Guide* for information about the `COMPATIBLE` initialization parameter

Topics:

22.8.1 Dependent Transactions

In the context of Flashback Transaction, transaction 2 can depend on transaction 1 in any of these ways:

- **Write-after-write dependency**

  Transaction 1 changes a row of a table, and later transaction 2 changes the same row.

- **Primary key dependency**

  A table has a primary key constraint on column c. In a row of the table, column c has the value v. Transaction 1 deletes that row, and later transaction 2 inserts a row into the same table, assigning the value v to column c.

- **Foreign key dependency**

  In table b, column b1 has a foreign key constraint on column a1 of table a. Transaction 1 changes a value in a1, and later transaction 2 changes a value in b1.

22.8.2 TRANSACTION_BACKOUT Parameters

The parameters of the `TRANSACTION_BACKOUT` procedure are:

- Number of transactions to be backed out
- List of transactions to be backed out, identified either by name or by XID
- Time hint, if you identify transactions by name

  Specify a time that is earlier than any transaction started.

- Backout option from Table 22-2

Table 22-2 Flashback TRANSACTION_BACKOUT Options

Option	Description
CASCADE	Backs out specified transactions and all dependent transactions in a post-order fashion (that is, children are backed out before parents are backed out). Without CASCADE, if any dependent transaction is not specified, an error occurs.
NOCASCADE	Default. Backs out specified transactions, which are expected to have no dependent transactions. First dependent transactions causes an error and appears in *_FLASHBACK_TXN_REPORT.
NOCASCADE_FORCE	Backs out specified transactions, ignoring dependent transactions. Server runs undo SQL statements for specified transactions in reverse order of commit times. If no constraints break and you are satisfied with the result, you can commit the changes; otherwise, you can roll them back.
NONCONFLICT_ONLY	Backs out changes to nonconflicting rows of the specified transactions. Database remains consistent, but transaction atomicity is lost.

TRANSACTION_BACKOUT analyzes the transactional dependencies, performs DML operations, and generates reports. TRANSACTION_BACKOUT does not commit the DML operations that it performs as part of transaction backout, but it holds all the required locks on rows and tables in the right form, preventing other dependencies from entering the system. To make the transaction backout permanent, you must explicitly commit the transaction.

> **See Also:**
>
> *Oracle Database PL/SQL Packages and Types Reference* for syntax of the TRANSACTION_BACKOUT procedure and detailed parameter descriptions

22.8.3 TRANSACTION_BACKOUT Reports

To see the reports that TRANSACTION_BACKOUT generates, query the static data dictionary views *_FLASHBACK_TXN_STATE and *_FLASHBACK_TXN_REPORT.

22.8.3.1 *_FLASHBACK_TXN_STATE

The static data dictionary view *_FLASHBACK_TXN_STATE shows whether a transaction is active or backed out. If a transaction appears in this view, it is backed out.

*_FLASHBACK_TXN_STATE is maintained atomically for compensating transactions. If a compensating transaction is backed out, all changes that it made are also backed out, and *_FLASHBACK_TXN_STATE reflects this. For example, if compensating transaction ct backs out transactions t1 and t2, then t1 and t2 appear in *_FLASHBACK_TXN_STATE. If ct itself is later backed out, the effects of t1 and t2 are reinstated, and t1 and t2 disappear from *_FLASHBACK_TXN_STATE.

> **See Also:**
>
> *Oracle Database Reference* for more information about `*_FLASHBACK_TXN_STATE`

22.8.3.2 *_FLASHBACK_TXN_REPORT

The static data dictionary view `*_FLASHBACK_TXN_REPORT` provides a detailed report for each backed-out transaction.

> **See Also:**
>
> *Oracle Database Reference* for more information about `*_FLASHBACK_TXN_REPORT`

22.9 Using Flashback Time Travel

Flashback Time Travel provides the ability to track and store definitional (including schema) and transactional changes to a table over its lifetime.

Using Flashback Time Travel, you can enable tracking of DML (such as `INSERT` and `DELETE`) and DDL operations (such as creating and truncating tables) on a table that is being tracked (also called tracked table). You can then use Flashback Data Archive (also called Flashback Archive) to archive the changes made to the rows of the tracked table in history tables. Flashback Time Travel also maintains a history of the evolution of a table schema. Having the history of the table and schema enables you to issue flashback queries (`AS OF` and `VERSIONS`) on the table and its schema. You can also view the history of DDL and DML changes made to the table.

With the Flashback Time Travel feature, you can create several Flashback Archives in your database. A Flashback Archive is a logical entity that is associated with a set of tablespaces. There is a quota that is reserved for the archives on those tablespaces and a retention period for the archived data. Using a Flashback Archive improves performance and helps in complying with record stage policies and audit reports.

A Flashback Archive consists of one or more tablespaces or parts thereof. You can have multiple Flashback Archives. If you are logged on as `SYSDBA`, you can specify a default Flashback Archive for the system. A Flashback Archive is configured with retention time. Data archived in the Flashback Archive is retained for the retention time that is specified when creating the Flashback Archive.

When choosing a Flashback Archive for a specific table, consider the data retention requirements for the table and the retention times of the Flashback Archives on which you have the `FLASHBACK ARCHIVE` object privilege.

To ensure data security of the Flashback Archive history tables, when enabling a Flashback Archive for a table, you can specify that a tracked table should use a blockchain history table for the Flashback Archive.

> **See Also:**
>
> Protecting Flashback Archive Data for more information about using blockchain history tables for the Flashback Archive

By default, Flashback Archive is not enabled for any table. Consider enabling Flashback Archive for user context tracking and database hardening.

- **User context tracking**. The metadata information for tracking transactions can include (if the feature is enabled) the user context, which makes it easier to determine which user made which changes to a table.

  To set the user context level (determining how much user context is to be saved), use the `DBMS_FLASHBACK_ARCHIVE.SET_CONTEXT_LEVEL` procedure. To access the context information, use the `DBMS_FLASHBACK_ARCHIVE.GET_SYS_CONTEXT` function.

- **Database hardening**. You can associate a set of tables together in an "application", and then enable Flashback Archive on all those tables with a single command. Database hardening also enables you to lock all the tables with a single command, preventing any DML on those tables until they are subsequently unlocked. Database hardening is designed to make it easier to use Flashback Time Travel to track and protect the security-sensitive tables for an application.

  To register an application for database hardening, use the `DBMS_FLASHBACK_ARCHIVE.REGISTER_APPLICATION` procedure, which is described in *Oracle Database PL/SQL Packages and Types Reference*.

You can also use Flashback Time Travel in various scenarios, such as enforcing digital shredding, accessing historical data, selective data recovery, and auditing.

Flashback Time Travel Restrictions

- You cannot enable Flashback Archive on tables with `LONG` data type or nested table columns.
- You cannot enable Flashback Archive on a nested table, temporary table, external table, materialized view, Advanced Query (AQ) table, hybrid partitioned tables, or non-table object.
- Flashback Archive does not support DDL statements that move, split, merge, or coalesce partitions or sub partitions, move tables, or convert `LONG` columns to LOB columns.
- Adding or enabling a Constraint (including Foreign Key Constraint) on a table that has been enabled for Flashback Archive fails with ORA-55610. Dropping or disabling a Constraint (including Foreign Key Constraint) on a table that has been enabled for Flashback Archive is supported.
- After enabling Flashback Archive on a table, Oracle recommends initially waiting at least 20 seconds before inserting data into the table and waiting up to 5 minutes before using Flashback Query on the table.
- Dropping a Flashback Archive base table requires Flashback Archive on the base table to be disabled first, and then the base table can be dropped. Disabling Flashback Archive will remove the historical data, while disassociating the Flashback Archive will retain the historical data. Truncate of the base table, on the other hand, is supported and the historical data will remain available in the Flashback Archive.
- If you enable Flashback Archive on a table, but Automatic Undo Management (AUM) is disabled, error ORA-55614 occurs when you try to modify the table.

- You cannot enable Flashback Archive if the table use any of these Flashback Time Travel reserved words as column names: `STARTSCN`, `ENDSCN`, `RID`, `XID`, `OP`, `OPERATION`.
- You can expect the Flashback Archive operations to slow down or become unresponsive when its space usage exceeds 90% of the maximum space for each of the available tablespaces that it is associated with. Ensure that you increase the maximum space that the Flashback Archive can use if the Flashback Archive usage nears 90% threshold for all associated tablespaces.
- In the standard mode (`MAX_COLUMNS = STANDARD`), a table that is enabled for Flashback Archive allows a maximum of 991 columns as apposed to a maximum of 1024 columns that are allowed for non-archived tables.
- In the extended mode (`MAX_COLUMNS = EXTENDED`), a table that is enabled for Flashback Archive allows a maximum of 4087 columns as opposed to a maximum of 4096 columns that are allowed for non-archived tables.

Topics:

> **See Also:**
>
> *Oracle Database PL/SQL Packages and Types Reference* for more information about `DBMS_FLASHBACK_ARCHIVE` package

22.9.1 DDL Statements on Tables Enabled for Flashback Archive

Flashback Archive supports only these DDL statements:

- `ALTER TABLE` statement that does any of the following:
  - Adds, drops, renames, or modifies a column
  - Adds, drops, or renames a constraint
  - Drops or truncates a partition or subpartition operation
- `TRUNCATE TABLE` statement
- `RENAME` statement that renames a table

Flashback Archive does not support DDL statements that move, split, merge, or coalesce partitions or subpartitions, move tables, or convert `LONG` columns to `LOB` columns.

For example, the following DDL statements cause error ORA-55610 when used on a table enabled for Flashback Archive:

- `ALTER TABLE` statement that includes an `UPGRADE TABLE` clause, with or without an `INCLUDING DATA` clause
- `ALTER TABLE` statement that moves or exchanges a partition or subpartition operation
- `DROP TABLE` statement

If you must use unsupported DDL statements on a table enabled for Flashback Archive, use the `DBMS_FLASHBACK_ARCHIVE.DISASSOCIATE_FBA` procedure to disassociate the base table from its Flashback Archive. To reassociate the Flashback Archive with the base table afterward, use the `DBMS_FLASHBACK_ARCHIVE.REASSOCIATE_FBA` procedure. Also, to drop a table enabled for Flashback Archive, you must first disable Flashback Archive on the table by using the `ALTER TABLE ... NO FLASHBACK ARCHIVE` clause.

See Also:

- *Oracle Database SQL Language Reference* for information about the `ALTER TABLE` statement
- *Oracle Database SQL Language Reference* for information about the `TRUNCATE TABLE` statement
- *Oracle Database SQL Language Reference* for information about the `RENAME` statement
- *Oracle Database SQL Language Reference* for information about the `DROP TABLE` statement
- *Oracle Database PL/SQL Packages and Types Reference* for information about the `DBMS_FLASHBACK_ARCHIVE` package

22.9.2 Creating a Flashback Archive

You can create a Flashback Archive with the `CREATE FLASHBACK ARCHIVE` statement.

Create a Flashback Archive with the `CREATE FLASHBACK ARCHIVE` statement, specifying:

- Name of the Flashback Archive
- Name of the first tablespace of the Flashback Archive
- (Optional) Maximum amount of space that the Flashback Archive can use in the first tablespace

  The default is unlimited. Unless your space quota on the first tablespace is also unlimited, you must specify this value; otherwise, error ORA-55621 occurs.

> **Note:**
>
> Ensure that you provide enough space as the maximum space for the Flashback Archive. If the used space exceeds 90% of the maximum space for each of the available Flashback Archive tablespaces, the Flashback Archive operations can slow down or become unresponsive.

- Retention time (number of days that Flashback Archive data for the table is guaranteed to be stored)
- (Optional) Whether to optimize the storage of data in the history tables maintained in the Flashback Archive, using `[NO] OPTIMIZE DATA`.

  The default is `NO OPTIMIZE DATA`.

If you are logged on as `SYSDBA`, you can also specify that this is the default Flashback Archive for the system. If you omit this option, you can still make this Flashback Archive as default at a later stage.

Oracle recommends that all users who must use Flashback Archive have unlimited quota on the Flashback Archive tablespace; however, if this is not the case, you must grant sufficient quota on that tablespace to those users.

Examples

- Create a default Flashback Archive named `fla1` that uses up to 10 G of tablespace `tbs1`, whose data are retained for one year:

```
CREATE FLASHBACK ARCHIVE DEFAULT fla1 TABLESPACE tbs1
  QUOTA 10G RETENTION 1 YEAR;
```

- Create a Flashback Archive named `fla2` that uses tablespace `tbs2`, whose data are retained for two years:

```
CREATE FLASHBACK ARCHIVE fla2 TABLESPACE tbs2 RETENTION 2 YEAR;
```

> **See Also:**
>
> - *Oracle Database SQL Language Reference* for more information about the `CREATE FLASHBACK ARCHIVE` statement syntax
> - Specifying the Default Flashback Archive

22.9.3 Altering a Flashback Archive

You can modify a Flashback Archive using the `ALTER FLASHBACK ARCHIVE` statement.

With the `ALTER FLASHBACK ARCHIVE` statement, you can:

- Change the retention time of a Flashback Archive
- Purge some or all of its data
- Add, modify, and remove tablespaces

> **Note:**
>
> Removing all tablespaces of a Flashback Archive causes an error.

If you are logged on as `SYSDBA`, you can also use the `ALTER FLASHBACK ARCHIVE` statement to make a specific file the default Flashback Archive for the system.

Examples

- Make Flashback Archive `fla1` the default Flashback Archive:

```
ALTER FLASHBACK ARCHIVE fla1 SET DEFAULT;
```

- To Flashback Archive `fla1`, add up to 5 G of tablespace `tbs3`:

```
ALTER FLASHBACK ARCHIVE fla1 ADD TABLESPACE tbs3 QUOTA 5G;
```

- To Flashback Archive `fla1`, add as much of tablespace `tbs4` as needed:

```
ALTER FLASHBACK ARCHIVE fla1 ADD TABLESPACE tbs4;
```

- Change the maximum space that Flashback Archive `fla1` can use in tablespace `tbs3` to 20 G:

```
ALTER FLASHBACK ARCHIVE fla1 MODIFY TABLESPACE tbs3 QUOTA 20G;
```

- Allow Flashback Archive `fla1` to use as much of tablespace `tbs1` as needed:

```
ALTER FLASHBACK ARCHIVE fla1 MODIFY TABLESPACE tbs1;
```

- Change the retention time for Flashback Archive `fla1` to two years:

```
ALTER FLASHBACK ARCHIVE fla1 MODIFY RETENTION 2 YEAR;
```

- Remove tablespace `tbs2` from Flashback Archive `fla1`:

```
ALTER FLASHBACK ARCHIVE fla1 REMOVE TABLESPACE tbs2;
```

  (Tablespace `tbs2` is not dropped.)

- Purge all historical data from Flashback Archive `fla1`:

```
ALTER FLASHBACK ARCHIVE fla1 PURGE ALL;
```

- Purge all historical data older than one day from Flashback Archive `fla1`:

```
ALTER FLASHBACK ARCHIVE fla1
  PURGE BEFORE TIMESTAMP (SYSTIMESTAMP - INTERVAL '1' DAY);
```

- Purge all historical data older than SCN 728969 from Flashback Archive `fla1`:

```
ALTER FLASHBACK ARCHIVE fla1 PURGE BEFORE SCN 728969;
```

> **See Also:**
>
> *Oracle Database SQL Language Reference* for more information about the `ALTER FLASHBACK ARCHIVE` statement

22.9.4 Dropping a Flashback Archive

You can drop a Flashback Archive with the `DROP FLASHBACK ARCHIVE` statement.

Dropping a Flashback Archive deletes its historical data, but does not drop its tablespaces.

Example

Remove Flashback Archive `fla1` and all its historical data, but not its tablespaces:

```
DROP FLASHBACK ARCHIVE fla1;
```

Oracle Database SQL Language Reference for more information about the `DROP FLASHBACK ARCHIVE` statement syntax

22.9.5 Specifying the Default Flashback Archive

You can specify the default Flashback Archive using the `CREATE` or `ALTER FLASHBACK ARCHIVE` statements.

The default Flashback Archive for the system is the default Flashback Archive for every user who does not have their own default Flashback Archive.

By default, the system has no default Flashback Archive. If you are logged on as `SYSDBA`, you can specify default Flashback Archive in either of these ways:

- Specify the name of an existing Flashback Archive in the `SET DEFAULT` clause of the `ALTER FLASHBACK ARCHIVE` statement. For example:

  ```
  ALTER FLASHBACK ARCHIVE fla1 SET DEFAULT;
  ```

  If `fla1` does not exist, an error occurs.

- Include `DEFAULT` in the `CREATE FLASHBACK ARCHIVE` statement when you create a Flashback Archive. For example:

  ```
  CREATE FLASHBACK ARCHIVE DEFAULT fla2 TABLESPACE tbs1
    QUOTA 10G RETENTION 1 YEAR;
  ```

> **See Also:**
>
> - *Oracle Database SQL Language Reference* for more information about the `CREATE FLASHBACK ARCHIVE` statement
> - *Oracle Database SQL Language Reference* for more information about the `ALTER DATABASE` statement

22.9.6 Enabling and Disabling Flashback Archive

By default, Flashback Archive is disabled for all table. You can enable Flashback Archive for a table if you have the `FLASHBACK ARCHIVE` object privilege on the Flashback Archive to use for that table.

To enable Flashback Archive for a table, include the `FLASHBACK ARCHIVE` clause in either the `CREATE TABLE` or `ALTER TABLE` statement. In the `FLASHBACK ARCHIVE` clause, you can specify the Flashback Archive where the historical data for the table are stored. The default is the default Flashback Archive for the system. If you specify a nonexistent Flashback Archive, an error occurs.

If a table has Flashback Archive enabled, and you try to enable it again with a different Flashback Archive, an error occurs.

After Flashback Archive is enabled for a table, you can disable it only if you either have the `FLASHBACK ARCHIVE ADMINISTER` system privilege or you are logged on as `SYSDBA`. To disable Flashback Archive for a table, specify `NO FLASHBACK ARCHIVE` in the `ALTER TABLE` statement. (It is unnecessary to specify `NO FLASHBACK ARCHIVE` in the `CREATE TABLE` statement, because that is the default.)

> **See Also:**
>
> *Oracle Database SQL Language Reference* for more information about the `FLASHBACK ARCHIVE` clause of the `CREATE TABLE` statement, including restrictions on its use

Examples

- Create table `employee` and store the historical data in the default Flashback Archive:

```
CREATE TABLE employee (EMPNO NUMBER(4) NOT NULL, ENAME VARCHAR2(10),
  JOB VARCHAR2(9), MGR NUMBER(4)) FLASHBACK ARCHIVE;
```

- Create table `employee` and store the historical data in the Flashback Archive `fla1`:

```
CREATE TABLE employee (EMPNO NUMBER(4) NOT NULL, ENAME VARCHAR2(10),
  JOB VARCHAR2(9), MGR NUMBER(4)) FLASHBACK ARCHIVE fla1;
```

- Enable Flashback Archive for the table `employee` and store the historical data in the default Flashback Archive:

```
ALTER TABLE employee FLASHBACK ARCHIVE;
```

- Enable Flashback Archive for the table `employee` and store the historical data in the Flashback Archive `fla1`:

```
ALTER TABLE employee FLASHBACK ARCHIVE fla1;
```

- Disable Flashback Archive for the table `employee`:

```
ALTER TABLE employee NO FLASHBACK ARCHIVE;
```

22.9.7 Viewing Flashback Archive Data

You can view information about Flashback Archive files in static data dictionary views.

Table 22-3 Static Data Dictionary Views for Flashback Archive Files

View	Description
`*_FLASHBACK_ARCHIVE`	Displays information about Flashback Archive files
`*_FLASHBACK_ARCHIVE_TS`	Displays tablespaces of Flashback Archive files
`*_FLASHBACK_ARCHIVE_TABLES`	Displays information about tables that are enabled for Flashback Archive

See Also:

- *Oracle Database Reference* for detailed information about `*_FLASHBACK_ARCHIVE`
- *Oracle Database Reference* for detailed information about `*_FLASHBACK_ARCHIVE_TS`
- *Oracle Database Reference* for detailed information about `*_FLASHBACK_ARCHIVE_TABLES`

22.9.8 Transporting Flashback Archive Data between Databases

You can export and import the Flashback Archive base tables along with their history to another database using the `FLASHBACK_ARCHIVE_MIGRATE` package and the Oracle Transportable Tablespaces capability.

DBMS_FLASHBACK_ARCHIVE_MIGRATE enables the migration of Flashback Archive enabled tables from a database on any release in which the package exists to any database on any release that supports Flashback Time Travel.

See Also:

Oracle Database PL/SQL Packages and Types Reference for more information about using the DBMS_FLASHBACK_ARCHIVE_MIGRATE package

22.9.9 Flashback Time Travel Scenarios

- Scenario: Using Flashback Time Travel to Enforce Digital Shredding
- Scenario: Using Flashback Time Travel to Access Historical Data
- Scenario: Using Flashback Time Travel to Generate Reports
- Scenario: Using Flashback Time Travel for Auditing
- Scenario: Using Flashback Time Travel to Recover Data

22.9.9.1 Scenario: Using Flashback Time Travel to Enforce Digital Shredding

Your company wants to "shred" (delete) historical data changes to the `Taxes` table after ten years. When you create the Flashback Archive for `Taxes`, you specify a retention time of ten years:

```
CREATE FLASHBACK ARCHIVE taxes_archive TABLESPACE tbs1 RETENTION 10 YEAR;
```

When history data from transactions on `Taxes` exceeds the age of ten years, it is purged. (The `Taxes` table itself, and history data from transactions less than ten years old, are not purged.)

22.9.9.2 Scenario: Using Flashback Time Travel to Access Historical Data

You want to be able to retrieve the inventory of all items at the beginning of the year from the table `inventory`, and to be able to retrieve the stock price for each symbol in your portfolio at the close of business on any specified day of the year from the table `stock_data`.

Create a default Flashback Archive named `fla1` that uses up to 10 G of tablespace `tbs1`, whose data are retained for five years (you must be logged on as `SYSDBA`):

```
CREATE FLASHBACK ARCHIVE DEFAULT fla1 TABLESPACE tbs1
  QUOTA 10G RETENTION 5 YEAR;
```

Enable Flashback Archive for the tables `inventory` and `stock_data`, and store the historical data in the default Flashback Archive:

```
ALTER TABLE inventory FLASHBACK ARCHIVE;
ALTER TABLE stock_data FLASHBACK ARCHIVE;
```

To retrieve the inventory of all items at the beginning of the year 2007, use this query:

```
SELECT product_number, product_name, count FROM inventory AS OF
  TIMESTAMP TO_TIMESTAMP ('2007-01-01 00:00:00', 'YYYY-MM-DD HH24:MI:SS');
```

To retrieve the stock price for each symbol in your portfolio at the close of business on July 23, 2007, use this query:

```
SELECT symbol, stock_price FROM stock_data AS OF
  TIMESTAMP TO_TIMESTAMP ('2007-07-23 16:00:00', 'YYYY-MM-DD HH24:MI:SS')
  WHERE symbol IN my_portfolio;
```

22.9.9.3 Scenario: Using Flashback Time Travel to Generate Reports

You want users to be able to generate reports from the table `investments`, for data stored in the past five years.

Create a default Flashback Archive named `fla2` that uses up to 20 G of tablespace `tbs1`, whose data are retained for five years (you must be logged on as `SYSDBA`):

```
CREATE FLASHBACK ARCHIVE DEFAULT fla2 TABLESPACE tbs1
  QUOTA 20G RETENTION 5 YEAR;
```

Enable Flashback Archive for the table `investments`, and store the historical data in the default Flashback Archive:

```
ALTER TABLE investments FLASHBACK ARCHIVE;
```

Lisa wants a report on the performance of her investments at the close of business on December 31, 2006. She uses this query:

```
SELECT * FROM investments AS OF
  TIMESTAMP TO_TIMESTAMP ('2006-12-31 16:00:00', 'YYYY-MM-DD HH24:MI:SS')
  WHERE name = 'LISA';
```

22.9.9.4 Scenario: Using Flashback Time Travel for Auditing

A medical insurance company must audit a medical clinic. The medical insurance company has its claims in the table `Billings`, and creates a default Flashback Archive named `fla4` that uses up to 100 G of tablespace `tbs1`, whose data are retained for 10 years:

```
CREATE FLASHBACK ARCHIVE DEFAULT fla4 TABLESPACE tbs1
  QUOTA 100G RETENTION 10 YEAR;
```

The company enables Flashback Archive for the table `Billings`, and stores the historical data in the default Flashback Archive:

```
ALTER TABLE Billings FLASHBACK ARCHIVE;
```

On May 1, 2007, clients were charged the wrong amounts for some diagnoses and tests. To see the records as of May 1, 2007, the company uses this query:

```
SELECT date_billed, amount_billed, patient_name, claim_Id,
  test_costs, diagnosis FROM Billings AS OF TIMESTAMP
  TO_TIMESTAMP('2007-05-01 00:00:00', 'YYYY-MM-DD HH24:MI:SS');
```

22.9.9.5 Scenario: Using Flashback Time Travel to Recover Data

An end user recovers from erroneous transactions that were previously committed in the database. The undo data for the erroneous transactions is no longer available, but because the required historical information is available in the Flashback Archive, Flashback Query works seamlessly.

Lisa manages a software development group whose product sales are doing well. On November 3, 2007, she decides to give all her level-three employees who have more than two years of experience a salary increase of 10% and a promotion to level four. Lisa asks her HR representative, Bob, to make the changes.

Using the HR web application, Bob updates the `employee` table to give Lisa's level-three employees a 10% raise and a promotion to level four. Then Bob finishes his work for the day and leaves for home, unaware that he omitted the requirement of two years of experience in his transaction. A few days later, Lisa checks to see if Bob has done the updates and finds that everyone in the group was given a raise! She calls Bob immediately and asks him to correct the error.

At first, Bob thinks he cannot return the employee table to its prior state without going to the backups. Then he remembers that the `employee` table has Flashback Archive enabled.

First, he verifies that no other transaction modified the `employee` table after his: The commit time stamp from the transaction query corresponds to Bob's transaction, two days ago.

Next, Bob uses these statements to return the `employee` table to the way it was before his erroneous change:

```
DELETE EMPLOYEE WHERE MANAGER = 'LISA JOHNSON';
INSERT INTO EMPLOYEE
  SELECT * FROM EMPLOYEE
```

```
    AS OF TIMESTAMP (SYSTIMESTAMP - INTERVAL '2' DAY)
      WHERE MANAGER = 'LISA JOHNSON';
```

Bob then runs again the update that Lisa had requested.

22.9.10 Protecting Flashback Archive Data

You can choose to protect the Flashback Archive (FBA) data for a tracked table (a user table whose changes need tracking). To do so, when creating a table (non-blockchain, non-immutable table), use the optional `BLOCKCHAIN` keyword with the `FLASHBACK ARCHIVE` clause. The archived data is then stored in an Oracle-managed blockchain log history table.

The following example code uses the `BLOCKCHAIN FLASHBACK ARCHIVE` clause with the `CREATE TABLE` command to create a table that is tracked using blockchain-based FBA.

```
CREATE TABLE tab1 (
  id           NUMBER,
  description  VARCHAR2(50),
  CONSTRAINT tab_1_pk PRIMARY KEY (id)
) BLOCKCHAIN FLASHBACK ARCHIVE fba_1year;
```

> **Note:**
>
> Ensure that the FBA that you specify already exists before creating the table.

> **See Also:**
>
> CREATE TABLE *flashback_archive_clause* and CREATE FLASHBACK ARCHIVE in *Oracle Database SQL Language Reference* for more information about designating a table as a tracked table for FBA

Having an FBA history table as a blockchain table enables you to record the history of changes made to regular user tables (that are tracked) in a cryptographically secure and verifiable manner. You can query the FBA history tables to determine if there is any tampering of the tracked table's content.

Blockchain Log History Table Overview

Blockchain Log history tables, like any other blockchain tables, are append-only tables that organize rows into a number of chains, with each row in a chain (except the first row) chained to the previous row in the chain using a cryptographic hash. Each change in a tracked table is added to the blockchain table as a separate row within the cryptographic hash chain that is created and maintained in the blockchain table.

> **See Also:**
>
> Managing BlockchainTables in *Oracle Database Administrator's Guide* for more information about blockchain tables

The following points summarize a blockchain log history table (blockchain history table).

- A blockchain history table is an Oracle-managed, append-only table with an Oracle-defined schema definition.
- A blockchain history table is used to track the changes in a user table and to ensure that the changes are tamper evident.
- You can verify the data and chain integrity of a blockchain history table using the standard blockchain table APIs.
- Blockchain tables have associated table-level and row-level retention periods. To determine the row retention period to use for the blockchain history tables, the higher of the row retention periods of the FBA history table and the blockchain table is used. Similarly, to determine the table retention period, the higher of the table retention periods of the FBA history table and the blockchain table is used.
- Row deletion in a blockchain history table is only allowed for a prefix of the chains.
- The owner of a blockchain history table is the same as that of the FBA history table.

Understanding How FBA Data is Stored

A row of a tracked table transitions through a series of lifespans, with each lifespan created whenever transactions commit changes to the row. The most recent lifespan (created by the last transaction that modified the row) is referred to as the "current lifespan," and all the previous lifespans are referred to as "historical lifespans."

Historical lifespans have a start System Change Number (SCN) and an end SCN, denoting the start and end time for the lifespan of a particular row version in a table. Current lifespans have a start SCN, but the end SCN is infinite. The end SCN of the newly created historical lifespan is set to the commit SCN of the latest transaction that affected the row.

Flashback Time Travel uses the following internal tables to record the lifespans from each tracked table:

- A blockchain history table (`SYS_FBA_HIST_<objno>`) to record the historical lifespans
- A temporary table (`SYS_FBA_TCRV_<objno>` or `tcrv` table) to record the current lifespan

When you enable FBA for a table, Flashback Time Travel internally creates the corresponding history and temporary (`tcrv`) tables. The blockchain history table stores all the historical row versions whenever a change happens on a row of the tracked table. The `tcrv` table (not a blockchain table) records the most recent row changes.

> **See Also:**
>
> Appendix: Recording DML Changes on the Tracked Table in *Appendices* for more information about how the DML updates to the tracked table are recorded in blockchain and temporary tables

Protecting FBA Data and Verifying its Integrity

Using blockchain tables for FBA history tables enables you to replay the secure history from the FBA to determine whether there has been any tampering in the tracked tables.

Blockchain works by computing and saving a cryptographic digest (blockchain digest) of the data. A blockchain digest is a well-defined summary of a blockchain table that is digitally signed by the Oracle instance using the table owner’s private key. Computing a blockchain

digest provides a snapshot of the metadata and data about the last row in each chain of a blockchain table at any particular time. This digest can be saved outside the database to detect tampering. If the data you are tracking has changed, then the blockchain digest of the new data differs from the previously stored digest.

> **See Also:**
>
> Format of Signed Digest in *Oracle Database Administrator's Guide* that describes the data format for the blockchain table digest

Blockchain tables are highly secure because they automatically chain new rows to existing rows cryptographically. For tables that are tracked using the `BLOCKCHAIN` keyword, as rows are added, a cryptographic digest is computed for the new row plus the digest of the previous row. Any modification to the data in the chain of rows breaks the cryptographic chain. To verify a blockchain, you can either use the standard blockchain verification procedures such as `DBMS_BLOCKCHAIN_TABLE.VERIFY_ROWS` from the database, or you can independently validate the blockchain outside the database.

All historical lifespans inserted in the blockchain history table are protected using the blockchain semantics and, therefore, can be verified for any tampering. You can use blockchain functions, such as `DBMS_BLOCKCHAIN_TABLE.GET_SIGNED_BLOCKCHAIN_DIGEST` to generate a signed blockchain digest for a specified blockchain table and verify the data integrity of any row changes.

For current lifespans, the relevant metadata is recorded in the `tcrv` table (`SYS_FBA_TCRV_<objno>`). You can use the `DBMS_FLASHBACK_ARCHIVE.GET_CURRENT_LIFESPAN_DIGEST` function to generate the current lifespan digest for the specified row in a tracked table. This API uses its associated metadata from the `tcrv` table to compute the current lifespan digest of the specific row of the tracked table. You can record this digest for detecting any corruptions in the row data.

Also, you can use the `DBMS_FLASHBACK_ARCHIVE.VERIFY_BLOCKCHAIN_LIFESPAN` procedure to verify current lifespans, historical lifespans, or all lifespans.

> **See Also:**
>
> - DBMS_BLOCKCHAIN_TABLE in *Oracle Database PL/SQL Language Reference* for detailed information about the blockchain digest function and verification procedures used for blockchain tables
> - DBMS_FLASHBACK_ARCHIVE in *Oracle Database PL/SQL Language Reference* for detailed information about the blockchain digest function and verification procedure for current lifespans

22.10 General Guidelines for Oracle Flashback Technology

Consider these best practices when using flashback technologies.

- Avoid issuing flashback queries on a table as of a time before Flashback Archive was enabled on the table. Alternatively, enable flashback archive before doing any DML operations on the table.

In this example, the sequence of actions will return the row with a=1. This may look like an incorrect result, and to avoid it, you must enable flashback archive before any row is inserted.

```
CREATE TABLE foo (a NUMBER);
VAR scn1 NUMBER;
BEGIN
   SELECT DBMS_FLASHBACK.GET_SYSTEM_CHANGE_NUMEBR INTO :scn FROM DUAL;
END;
INSERT INTO foo VALUES (1);
COMMIT;
ALTER TABLE foo FLASHBACK ARCHIVE;
UPDATE foo SET a=2 WHERE a=1;
COMMIT;
SELECT * FROM foo AS OF SCN :scn;
```

- Use the `DBMS_FLASHBACK.ENABLE` and `DBMS_FLASHBACK.DISABLE` procedures around SQL code that you do not control, or when you want to use the same past time for several consecutive queries.
- Use Oracle Flashback Query, Oracle Flashback Version Query, or Oracle Flashback Transaction Query for SQL code that you write, for convenience. An Oracle Flashback Query, for example, is flexible enough to do comparisons and store results in a single query.
- To obtain an SCN to use later with a flashback feature, use `DBMS_FLASHBACK.GET_SYSTEM_CHANGE_NUMBER`.
- To compute or retrieve a past time to use in a query, use a function return value as a time stamp or SCN argument. For example, add or subtract an `INTERVAL` value to the value of the `SYSTIMESTAMP` function.
- Use Oracle Flashback Query, Oracle Flashback Version Query, and Oracle Flashback Transaction Query locally or remotely. An example of a remote Oracle Flashback Query is:

  ```
  SELECT * FROM employees@some_remote_host AS OF
      TIMESTAMP (SYSTIMESTAMP - INTERVAL '60' MINUTE);
  ```

- To ensure database consistency, perform a `COMMIT` or `ROLLBACK` operation before querying past data.
- Remember that all flashback processing uses the current session settings, such as national language and character set, not the settings that were in effect at the time being queried.
- Remember that DDLs that alter the structure of a table (such as drop/modify column, move table, drop partition, truncate table/partition, and add constraint) invalidate any existing undo data for the table. If you try to retrieve data from a time before such a DDL executed, error ORA-01466 occurs. DDL operations that alter the storage attributes of a table (such as `PCTFREE`, `INITRANS`, and `MAXTRANS`) do not invalidate undo data.
- To query past data at a precise time, use an SCN. If you use a time stamp, the actual time queried might be up to 3 seconds earlier than the time you specify. Oracle Database uses SCNs internally and maps them to time stamps at a granularity of 3 seconds.

  For example, suppose that the SCN values 1000 and 1005 are mapped to the time stamps 8:41 AM and 8:46 AM, respectively. A query for a time between 8:41:00 and 8:45:59 AM is mapped to SCN 1000; an Oracle Flashback Query for 8:46 AM is mapped to SCN 1005. Therefore, if you specify a time that is slightly after a DDL operation (such as a table creation) Oracle Database might use an SCN that is immediately before the DDL operation, causing error ORA-01466.

- You cannot retrieve past data from a dynamic performance (`V$`) view. A query on such a view returns current data.
- You can perform queries on past data in static data dictionary views, such as `*_TABLES`.

> **Note:**
>
> If you cannot retrieve past data using a static data dictionary view, then you can query the corresponding base table to retrieve the data. However, Oracle does not recommend that you use the base tables directly because they are normalized and most data is stored in a cryptic format.

- You can enable optimization of data storage for history tables maintained by Flashback Archive by specifying `OPTIMIZE DATA` when creating or altering a Flashback Archive history table.

  `OPTIMIZE DATA` optimizes the storage of data in history tables by using any of these features:

  – Advanced Row Compression
  – Advanced LOB Compression
  – Advanced LOB Deduplication
  – Segment-level compression tiering
  – Row-level compression tiering

  The default is not to optimize the storage of data in history tables.
- After you create a VPD policy, consider creating an equivalent policy for the Flashback Archive history table.

  See *Oracle Database Security Guide* for more information about Virtual Private Database Policies and Flashback Time Travel.

> **Caution:**
>
> Importing user-generated history can lead to inaccurate, or unreliable results. This procedure should only be used after consulting with Oracle Support.

22.11 Oracle Virtual Private Database Policies and Oracle Flashback Time Travel

Oracle Virtual Private Database policies do not automatically work with Oracle Flashback Time Travel.

After you create an Oracle Virtual Private Database (VPD) policy for a table, consider creating an equivalent policy for the Flashback Archive history table. The following example demonstrates how to do so.

Example 22-3 Creating an Equivalent Policy for an Flashback Archive History Table

1. Create a temporary VPD administrative user.

```
CREATE USER sysadmin_vpd IDENTIFIED BY password CONTAINER = CURRENT;
GRANT CREATE SESSION, CREATE ANY CONTEXT, CREATE PROCEDURE TO sysadmin_vpd;
GRANT EXECUTE ON DBMS_SESSION TO sysadmin_vpd;
GRANT EXECUTE ON DBMS_FLASHBACK, DBMS_FLASHBACK_ARCHIVE TO sysadmin_vpd;
GRANT EXECUTE ON DBMS_RLS TO sysadmin_vpd;
GRANT UPDATE ON SCOTT.EMP TO sysadmin_vpd;
```

2. Connect to the PDB as the `sysadmin_vpd` user.

```
connect sysadmin_vpd@pdb_name
Enter password: password
Connected.
```

3. Create the VPD function.
   For example, the following function shows only rows with department number (`deptno`) 30 to users other than user `SCOTT`:

```
CREATE OR REPLACE FUNCTION emp_policy_func (
 v_schema IN VARCHAR2,
 v_objname IN VARCHAR2)

RETURN VARCHAR2 AS
condition VARCHAR2 (200);

BEGIN
 condition := 'deptno=30';
 IF sys_context('userenv', 'session_user') IN ('SCOTT') THEN
   RETURN NULL;
 ELSE
   RETURN (condition);
 END IF;
END emp_policy_func;
/
```

4. Create the following VPD procedure to attach the `emp_policy_func` function to the `SCOTT.EMP` table.

```
BEGIN
 DBMS_RLS.ADD_POLICY (
  object_schema     => 'scott',
  object_name       => 'emp',
  policy_name       => 'emp_policy',
  function_schema   => 'sysadmin_vpd',
  policy_function   => 'emp_policy_func',
  policy_type       =>  dbms_rls.dynamic);
END;
/
```

5. Create the following `test` user and grant privileges, including those related to Flashback Archive.

```
CREATE USER test IDENTIFIED BY password;
GRANT CREATE SESSION TO test;
GRANT CONNECT, RESOURCE TO test;
GRANT SELECT ON SCOTT.EMP TO test;
GRANT FLASHBACK ARCHIVE ON ftest TO test;
GRANT EXECUTE ON DBMS_FLASHBACK_ARCHIVE TO test;
GRANT EXECUTE ON DBMS_FLASHBACK TO test;
GRANT FLASHBACK ANY TABLE TO PUBLIC;
GRANT EXECUTE ON emp_policy_func TO PUBLIC;
```

6. Enable the `SCOTT.EMP` table for flashback archive, and for transactions

```
ALTER TABLE SCOTT.EMP FLASHBACK ARCHIVE;
```

7. Perform an update to the `SCOTT.EMP` table.

```
UPDATE SCOTT.EMP SET SAL=SAL+1;
COMMIT;
```

8. Put the preceding procedure to sleep for 60 seconds.

```
EXEC DBMS_LOCK.SLEEP(60);
```

9. Connect as user `test`.

```
connect test@pdb_name
Enter password: password
Connected.
```

10. Perform the following query to show only rows that have `deptno=30`, per the VPD policy:

```
SELECT EMPNO,DEPTNO,SAL FROM SCOTT.EMP;
```

    The VPD policy is not working because all rows are shown.

```
SELECT EMPNO,DEPTNO,SAL FROM SCOTT.EMP AS OF TIMESTAMP SYSDATE-1;
```

11. Connect as user `sysadmin_vpd`.

```
connect sysadmin_vpd@pdb_name
Enter password: password
Connected.
```

12. Find the object ID for the `EMP` table.

```
SELECT OBJECT_ID FROM DBA_OBJECTS WHERE OBJECT_NAME='EMP';
```

13. Define a similar VPD policy on the `SYS_FBA_HIST_object_id_of_EMP_table` table. This table is internally created by Flashback Archive

```
BEGIN
 DBMS_RLS.ADD_POLICY (
  object_schema     => 'scott',
  object_name       => 'sys_fba_hist_object_id_of_EMP_table',
  policy_name       => 'emp_hist_policy',
  function_schema   => 'sysadmin_vpd,
  policy_function   => 'emp_policy_func',
  policy_type       =>  dbms_rls.dynamic);
END;
/
```

14. Connect as the `test` user.

```
connect test@pdb_name
Enter password: password
Connected.
```

15. Test the policy again:

```
SELECT EMPNO,DEPTNO,SAL FROM SCOTT.EMP AS OF TIMESTAMP SYSDATE-1;
```

    Now the VPD policy works, because the query only shows rows with `deptno=30`.

16. Connect as a user who can drop user accounts.
    For example:

```
connect sec_admin@pdb_name
Enter password: password
Connected.
```

17. Drop the `sysadmin_vpd` user and its objects as follows:

```
DROP USER sysadmin_vpd CASCADE;
```

22.12 Performance Guidelines for Oracle Flashback Technology

- Use the `DBMS_STATS` package to generate statistics for all tables involved in an Oracle Flashback Query. Keep the statistics current. Oracle Flashback Query uses the cost-based optimizer, which relies on these statistics.
- Minimize the amount of undo data that must be accessed. Use queries to select small sets of past data using indexes, not to scan entire tables. If you must scan a full table, add a parallel hint to the query.

  The performance cost in I/O is the cost of paging in data and undo blocks that are not in the buffer cache. The performance cost in CPU use is the cost of applying undo information to affected data blocks. When operating on changes in the recent past, flashback operations are CPU-bound.

  Oracle recommends that you have enough buffer cache, so that the versions query for the archiver finds the undo data in the buffer cache. Buffer cache access is significantly faster than disk access.

- If very large transactions (such as affecting more than 1 million rows) are performed on tracked tables, set the large pool size high enough (at least 1 GB) for Parallel Query not to have to allocate new chunks out of the SGA.
- For Oracle Flashback Version Query, use index structures. Oracle Database keeps undo data for index changes and data changes. Performance of index lookup-based Oracle Flashback Version Query is an order of magnitude faster than the full table scans that are otherwise needed.
- In an Oracle Flashback Transaction Query, the `xid` column is of the type `RAW(8)`. To take advantage of the index built on the `xid` column, use the `HEXTORAW` conversion function: `HEXTORAW(xid)`.
- An Oracle Flashback Query against a materialized view does not take advantage of query rewrite optimization.

> **See Also:**
>
> *Oracle Database Performance Tuning Guide* for information about setting the large pool size

22.13 Multitenant Container Database Restrictions for Oracle Flashback Technology

These Oracle Flashback Technology features are unavailable for a multitenant container database (CDB):

- Flashback Transaction Query is not supported in a CDB in shared undo mode. It is only supported in local undo mode.
- Flashback Transaction Backout is not supported in a CDB.