Database Archiving and Temporal

# Using In-Database Row Archiving

## Creating a Table and Enabling Row Archival

Connect to pdb1 as the HR user.

**. oraenv  
[Enter cdb1 at the prompt.]**

**sqlplus hr/hr@localhost:1521/pdb1**

Create a copy of the HR.employees table. Call the copy HR.emp\_arch, and only copy over 4 rows.

**create table emp\_arch  
as select employee\_id, first\_name from employees where rownum <= 4;**

Enable row archival on HR.emp\_arch.

**alter table emp\_arch row archival;**

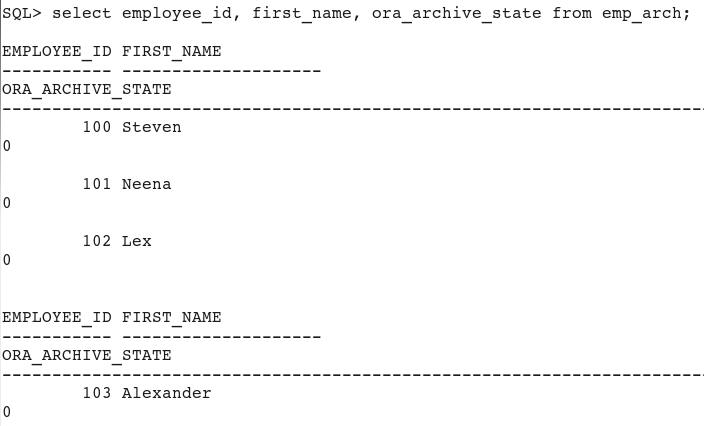
There is a hidden column in row-archival-enabled tables called ora\_archive\_state which indicates whether a row is archived or not. The hidden column is only displayed if specified in a query. First, describe the table structure ofHR.emp\_arch. Notice that the ora\_archive\_state column is **not listed**.

**desc emp\_arch**



Now, query the HR.emp\_arch table. Display the ora\_archive\_state column in the query result.

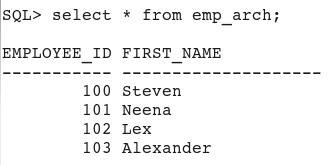
**select employee\_id, first\_name, ora\_archive\_state from emp\_arch;**



A value of 0 in the ora\_archive\_state column means the row is not archived; in other words, the row is active and visible via a standard query.

Confirm that all rows are active by issuing a standard query. You should see all 4 rows.

**select \* from emp\_arch;**



## Setting the Archive State of Some Rows

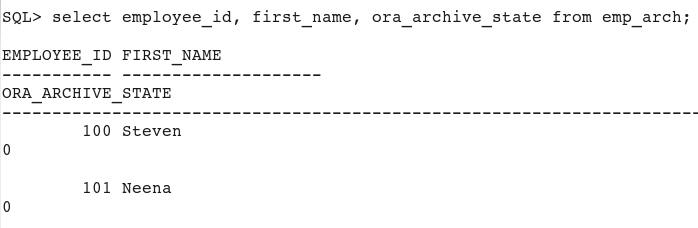
Use the dbms\_ilm.archivestatename procedure to update the ora\_archive\_state value for employee\_id 102 and 103.

**update emp\_arch  
set ora\_archive\_state=dbms\_ilm.archivestatename(1)  
where employee\_id in (102, 103);**

**commit;**

Confirm that now only 2 rows are visible via a standard query. Include the ora\_archive\_state column in the query. The ora\_archive\_state value for the 2 rows should be 0.

**select employee\_id, first\_name, ora\_archive\_state from emp\_arch;**



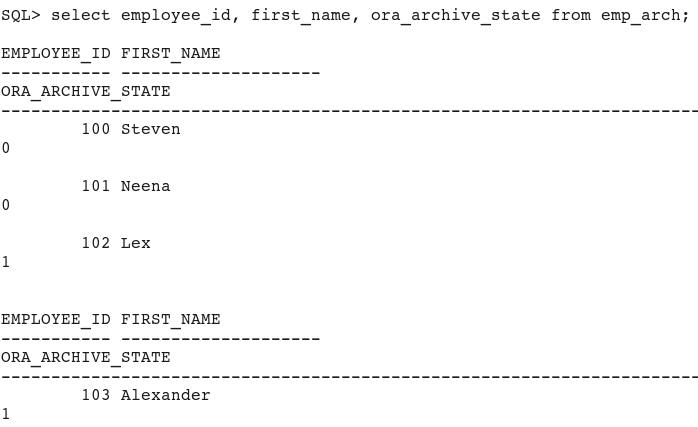
## Enabling a Session to View Archived Rows

Set the archival visibility to ALL for the session. This will allow this session to view all rows, archived or not.

**alter session set row archival visibility = all;**

Issue the same query as before, but now you should see all 4 rows.

**select employee\_id, first\_name, ora\_archive\_state from emp\_arch;**

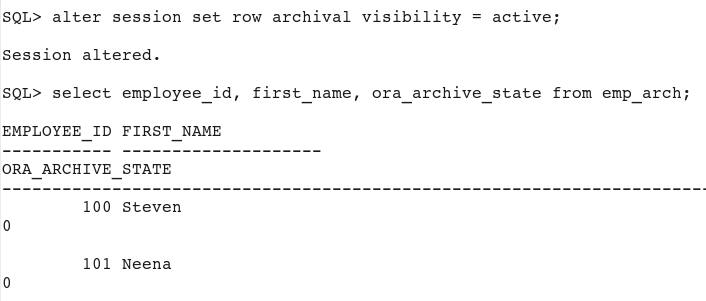


Set the archival visibility to ACTIVE for the session. This will allow this session to view only active (unarchived) rows.

**alter session set row archival visibility = active;**

Issue the same query as before, but now you should see only the 2 active rows.

**select employee\_id, first\_name, ora\_archive\_state from emp\_arch;**



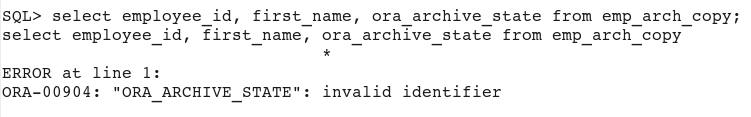
## Verifying that Copying a Row-Archival Table Will Not Propagate the Source Table's Archival State to the Target Table

Use CTAS to make a copy of the HR.emp\_arch table.

**create table emp\_arch\_copy as select employee\_id, first\_name from emp\_arch;**

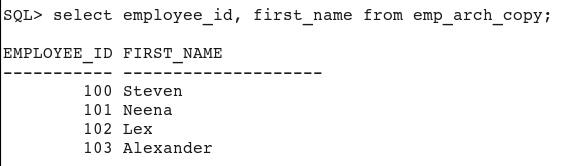
Verify that HR.emp\_arch\_copy does not have row archival enabled, and therefore does not contain the ora\_archive\_state hidden column. You should see an "invalid identifier" error for this query.

**select employee\_id, first\_name, ora\_archive\_state from emp\_arch\_copy;**



Query the table again, but this time omit the hidden column. Notice that all 4 rows are copied even though you set the row archival visibility for this session to active rows only.

**select employee\_id, first\_name from emp\_arch\_copy;**



Set the archival visibility to ALL for the session. From this point forward, this session should now see all rows, whether archived or not.

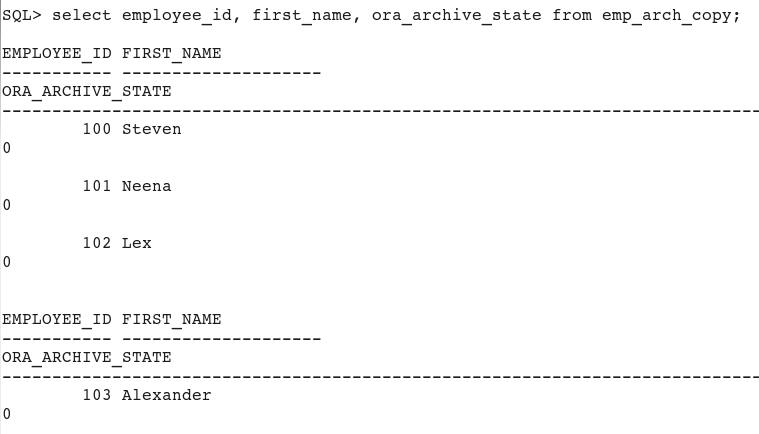
**alter session set row archival visibility = all;**

Enable row archival for HR.emp\_arch\_copy.

**alter table emp\_arch\_copy row archival;**

Verify that HR.emp\_arch\_copy now has row archival enabled, and therefore contains the ora\_archive\_state hidden column.

**select employee\_id, first\_name, ora\_archive\_state from emp\_arch\_copy;**



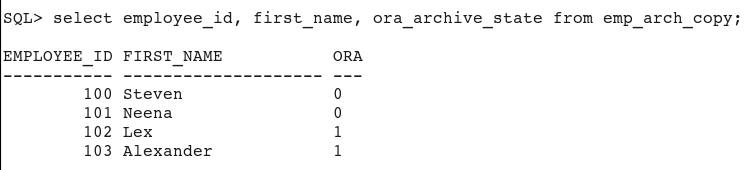
Use the dbms\_ilm.archivestatename procedure to update the ora\_archive\_state value for employee\_id 102 and 103.

**update emp\_arch\_copy  
set ora\_archive\_state=dbms\_ilm.archivestatename(1)  
where employee\_id in (102, 103);**

**commit;**

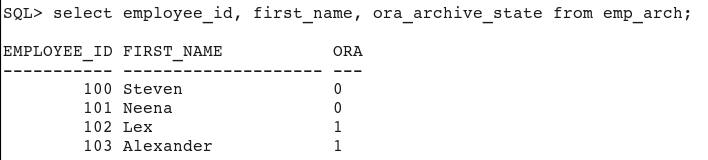
Confirm that now 2 rows are active, and 2 rows are inactive, via a standard query. Include the ora\_archive\_state column in the query. The ora\_archive\_state value for the 2 active rows should be 0, and the 2 inactive rows should be 1.

**select employee\_id, first\_name, ora\_archive\_state from emp\_arch\_copy;**



Recall that the HR.emp\_arch table you created earlier contains 2 active and 2 inactive rows. Confirm this via a standard query. Include the ora\_archive\_state column in the query. The ora\_archive\_state value for the 2 active rows should be 0, and the 2 inactive rows should be 1.

**select employee\_id, first\_name, ora\_archive\_state from emp\_arch;**



Copy all rows from HR.emp\_arch to HR.emp\_arch\_copy. To distinguish between rows that were previously copied to HR.emp\_arch\_copy via CTAS, and the rows being copied again now, append '\_New' to the values in thefirst\_name column values.

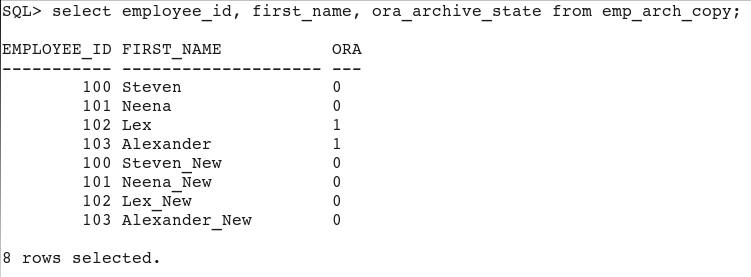
**insert into emp\_arch\_copy select employee\_id, first\_name || '\_New' from emp\_arch;**

**commit;**

What row archival state do you think the newly inserted rows have? Remember that the archival state is not copied along with the rows.

Query HR.emp\_arch\_copy to determine the ora\_archive\_state value for each row.

**select employee\_id, first\_name, ora\_archive\_state from emp\_arch\_copy;**



You should see that the ora\_archive\_state values from HR.emp\_arch are not propagated to HR.emp\_arch\_copy. All of the \_New rows in HR.emp\_arch\_copy have ora\_archive\_state set to the default value of 0 (active). The 2 inactive rows resulted from the update you performed in the HR.emp\_arch\_copy table in step 7.

## Resetting Your Environment

Perform the following steps to reset your environment prior to repeating the activities covered in this OBE or starting another OBE.

Drop the tables created in this tutorial.

**drop table emp\_arch;**

**drop table emp\_arch\_copy;**

# Implementing Temporal  Validity

\*\*\*\*Only support to non-cdb

## Creating a Table with a Valid-Time Dimension

Using SQL\*Plus, connect to noncdb database as sys.

**sqlplus system/oracle@localhost:1521/noncdb**

Create a copy of the HR.employees table. Call the new table HR.emp\_temp.

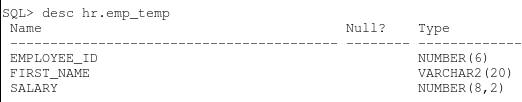
**create table hr.emp\_temp as   
select employee\_id, first\_name, salary   
from hr.employees  
where rownum <=10;**

Add a valid-time dimension to the HR.emp\_temp table.

**alter table hr.emp\_temp add period for valid\_time;**

View the HR.emp\_temp table structure.

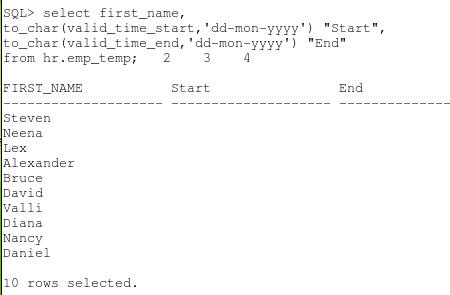
**desc hr.emp\_temp**



Notice that the valid-time columns are not displayed because they are hidden columns and must be explicitly included in a query.

Display all records in HR.emp\_temp, including the valid-time columns.

**select first\_name,  
to\_char(valid\_time\_start,'dd-mon-yyyy') "Start",  
to\_char(valid\_time\_end,'dd-mon-yyyy') "End"  
from hr.emp\_temp;**



Update the records and populate the valid-time columns.

**update hr.emp\_temp  
set valid\_time\_start = to\_date('01-JUN-1995','dd-MON-yyyy'), valid\_time\_end = to\_date('15-SEP-2010','dd-MON-yyyy')  
where first\_name in ('Lex','Alexander','Bruce','David','Daniel');**

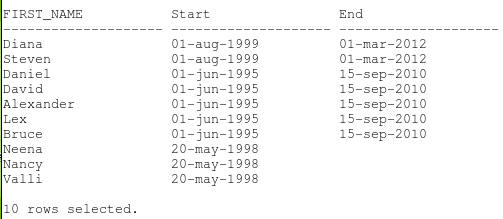
**update hr.emp\_temp  
set valid\_time\_start = to\_date('01-AUG-1999','dd-MON-yyyy'), valid\_time\_end = to\_date('01-MAR-2012','dd-MON-yyyy')  
where first\_name in ('Steven','Diana');**

**update hr.emp\_temp  
set valid\_time\_start = to\_date('20-MAY-1998','dd-MON-yyyy')  
where first\_name in ('Neena','Nancy','Valli');**

**commit;**

Display all records in HR.emp\_temp, including the valid-time columns.

**select first\_name,  
to\_char(valid\_time\_start,'dd-mon-yyyy') "Start",  
to\_char(valid\_time\_end,'dd-mon-yyyy') "End"  
from hr.emp\_temp  
order by 2;**

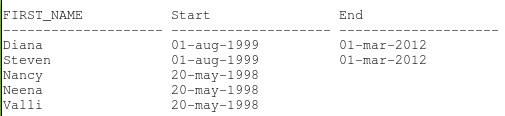


## Querying a Table with Valid Time Support

Here are some examples of queries on tables with valid time support.

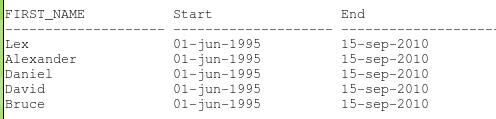
Find all employee records which are still valid as of 01-JUN-2011.

**select first\_name,  
to\_char(valid\_time\_start,'dd-mon-yyyy') "Start",  
to\_char(valid\_time\_end,'dd-mon-yyyy') "End"  
from hr.emp\_temp  
as of period for valid\_time to\_date('01-JUN-2011')  
order by 2;**



Find all employee records which are valid between 01-SEP-1995 and 01-SEP-1996. This query should return all records where valid\_time\_start >= 01-SEP-1995 and valid\_time\_end <= 01-SEP-1996.

**select first\_name,  
to\_char(valid\_time\_start,'dd-mon-yyyy') "Start",  
to\_char(valid\_time\_end,'dd-mon-yyyy') "End"   
from hr.emp\_temp   
versions period for valid\_time   
between to\_date('01-SEP-1995') and to\_date('01-SEP-1996')   
order by 2;**



## Setting Visibility of Temporal Data with DBMS\_FLASHBACK\_ARCHIVE.ENABLE\_AT\_VALID\_TIME

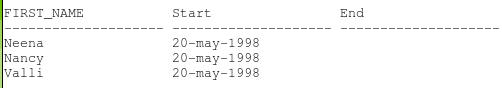
To set the visibility of data in tables with temporal support, you use the DBMS\_FLASHBACK\_ARCHIVE.enable\_at\_valid\_time procedure.

Only make visible the currently valid records in HR.emp\_temp.

**exec dbms\_flashback\_archive.enable\_at\_valid\_time('CURRENT');**

Query HR.emp\_temp. Which records are currently valid and will be displayed?

**select first\_name,  
to\_char(valid\_time\_start,'dd-mon-yyyy') "Start",  
to\_char(valid\_time\_end,'dd-mon-yyyy') "End"  
from hr.emp\_temp  
order by 2;**

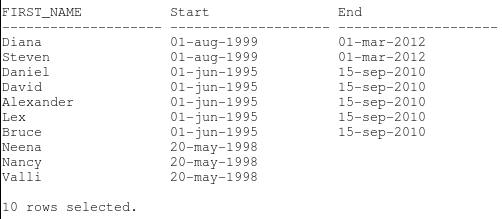


Make all records in HR.emp\_temp visible, regardless of their temporal validity status.

**exec dbms\_flashback\_archive.enable\_at\_valid\_time('ALL');**

Query HR.emp\_temp. All records should be displayed.

**select first\_name,  
to\_char(valid\_time\_start,'dd-mon-yyyy') "Start",  
to\_char(valid\_time\_end,'dd-mon-yyyy') "End"  
from hr.emp\_temp  
order by 2;**



## Resetting Your Environment

Perform the following steps to reset your environment prior to repeating the activities covered in this OBE or starting another OBE.

Drop the HR.emp\_temp table.

**drop table hr.emp\_temp purge;**