Flashback Table LAB

Let's take a look at the above scenario. The problematic year-end batch run might have affected only a few tables. For example, it may have updated only the table `EMP` with new COMMISSIONS (COMM). If this is the case, John can use the flashback table feature, which reinstates a table to a point in the past.

There is no special setup necessary to perform the flashback table operation. The only requirement is that the table must have row movement enabled—either at table creation time or later using the `ALTER TABLE ACCOUNTS ENABLE ROW MOVEMENT` statement. The `FLASHBACK TABLE` statement reads the past images of the table from the undo segments and reconstructs the table rows using the flashback queries introduced in Oracle9*i*.

If a non-DBA user other than the schema owner performs a flashback table operation, she needs `SELECT, DELETE, INSERT, ALTER`, and `FLASHBACK` privileges on that table or the equivalent `ANY TABLE` system privileges. First create a user called john with the same password and grant them connect, resource roles. We also need to allow for flashback table row movement. For the emp and dept tables we will alter them to allow for the row movement;

```
SQL> alter table emp enable row movement;
```

```
SQL> alter table dept enable row movement;
```

For John, the table `EMP` looks like this:

```
Name                   Null?        Type
---------------------- ------------ ------------
EMPNO                  NOT NULL     NUMBER(4)
ENAME                               CHAR(10)
JOB                                 CHAR(9)
MGR                                 NUMBER(4)
HIREDATE                            DATE
SAL                                 NUMBER(7,2)
COMM                                NUMBER(7,2)
DEPTNO                              NUMBER(2)
```

Update the emp table and set the sal for empno 7934 to 7000.00 monthly salary and commit the data. Note the time at which you do this.

SQL> update scott.emp

set sal = 7000.00

where emno=7934;

SQL> update scott.emp

set comm=500;

SQL> commit;

Here are the steps John takes to use the flashback table feature:

1. He asks for an approximate point in time to which the database has to be retraced; the answer is about 11:00 p.m.
2. He defines the desired logical reference point to go back to. Here's what he sees when he queries the table now:

```
   select ENAME,COMM
from EMP
WHERE EMPNO BETWEEN 7300 AND 7500;
     ENAME   COMM
        ------  --------


          SMITH

          ALLEN   300
```

3. The output shows that everyone has a COMM = 500 and that is inaccurate. The desired logical reference point should be where all the employees have commissions who had them before the update. To see the prior data

4. select to_char(sysdate,'YYYY-MM-DD HH:MI:SS') FROM dual to get the current time. Let's go back 10 minutes before the update and look at the data.

select * from emp as of timestamp to_timestamp('08-AUG-14','dd-MON-yy')

.

He issues this statement to restore the table `emp table` to that prior update time:

```
   flashback table ACCOUNTS to timestamp
13.  to_timestamp (<time before update,'mm/dd/yyyy
   hh24:mi:ss');
```

Voilà! The entire table is reconstructed as of that time stamp. John can flash back to a point as far in the past as the remaining undo data in undo segments allow. Instead of the time stamp, John can also use the system change number (SCN) as follows:

```
flashback table emp
to SCN 9988653338;
```

This method could employ finding a suitable SCN number far enough back to accomplish the above task. First we would query the V$FLASHBACK_DATABASE_LOG table for the oldest scn number available.

```
select oldest_flashback_scn, oldest_flashback_time
 from v$flashback_database_log;
OLDEST_FLASHBACK_SCN        OLDEST_FL
------------------------------------        ----------------
  2659752                                16-MAR-05
```

Second we would find the current SCN # as follows

```
select current_scn from v$database;
CURRENT_SCN
----------------------
   2663240
```

John could then flash back the table to a particular time 10:30 p.m.—and then rechecks the status:

```
14.  flashback table emp to timestamp
15.  to_timestamp ('12/31/2003 22:30:00','mm/dd/yyyy
   hh24:mi:ss');
```

Since the table is never dropped, all the dependent objects—such as indexes, constraints, triggers, and so on—remain intact. All independent objects referencing this table, such as procedures, also remain valid. Even global indexes on partitioned tables are maintained and remain valid.

If John wanted to flash back the table `dept owned by scott and the table emp and he had the privilege to do so`, he could have used the flashback table command as follows:

```
flashback table scott.dept, scott.emp to scn 1234567;
```

The entire flashback table operation is done through a single, powerful SQL statement.

Let's examine another situation. Suppose that Laura accidentally drops a key lookup table, scott.`dept`. Realizing the mistake, she asks John if he can reinstate the table. In prior versions of Oracle Database, this would have required a point-in-time recovery. In Oracle Database 12c, however, dropping a table renames the table and places it in a logical container known as the Recycle Bin.

To recover the table, John simply issues the following command:

```
flashback table scott.dept to before drop;
```

The table reappears instantly, without needing any kind of recovery. Note that unlike the flashback operation described earlier, this does not require data reconstruction through the undo segments; rather the table is merely moved back from the recycle bin.