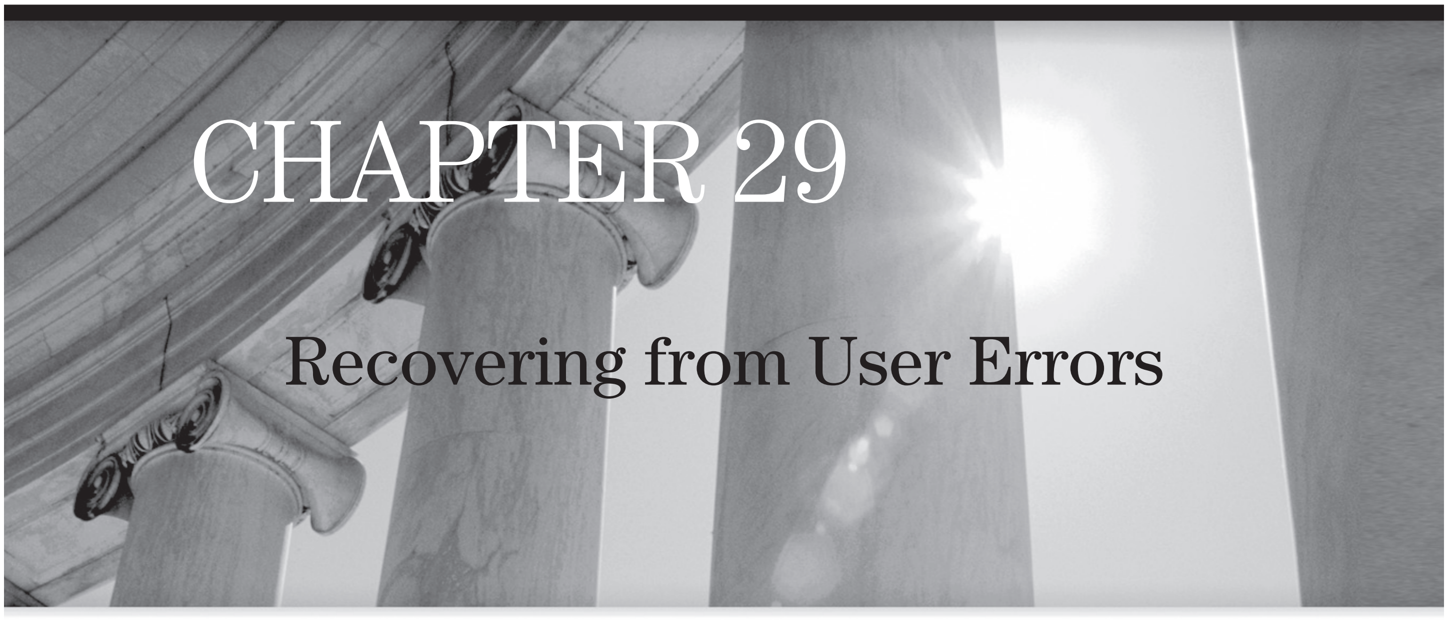


CHAPTER 29

Recovering from User Errors

In this chapter you will learn how to

- Recover a dropped table using Flashback technology
- Manage the recycle bin
- Perform a Flashback Table operation
- Recover from user errors using Flashback Versions Query
- Perform transaction-level recovery using Flashback Transaction Query

The previous chapter introduced Flashback Database, a powerful but drastic feature that is functionally equivalent to an incomplete recovery. This chapter covers the other flashback technologies available in an Oracle 10*g* database. These are not as extreme as Flashback Database in that they do not entail either downtime or loss of data. They are still, however, very powerful techniques for recovering from errors by backing out changes that you would prefer not to have been committed.

The flashback technologies discussed here are, first, Flashback Drop, enabled by the way the DROP TABLE command is used and implemented, and second, various ways of exploiting the UNDO capability: Flashback Versions Query, Flashback Table Query, and Flashback Transaction Query.

Flashback and the ACID Test

Remember the ACID test, described in Chapter 9. This is part of the rules to which a relational database must conform, and is critical to an understanding of the flashback technologies: both their capabilities and their limitations.

All DML transactions are terminated with either a COMMIT or a ROLLBACK statement. Until then, while the transaction is in progress, the principle of transaction isolation (the *I* of the ACID test) as implemented by Oracle guarantees that no one, other than the session carrying out the transaction, can see the changes it has made.

Furthermore, the principle of atomicity (the *A* of the ACID test) guarantees that the session can terminate the transaction with a ROLLBACK, which will reverse the changes completely; no other session will have any knowledge that the changes were ever made. If the transaction is terminated with a COMMIT, then the changes will be immediately visible to all other sessions. The only exception to this is any sessions that for reasons of read consistency (the *C* of the ACID test) need to be protected from the changes. Furthermore, once a transaction is committed, it must be absolutely impossible for the database to lose the changes; this is the *D*, for durable, of the ACID test.

In many ways, DDL commands are transactions like any other. The rules of a relational database require that the effect of committed DDL can never be reversed, and all DDL is committed, automatically. You have no control over this; the COMMIT is an integral part of all DDL commands.

Flashback Drop provides a means whereby you can reverse the effect of a DROP TABLE command, but there is no guarantee that it will succeed. This will depend on other activity in the database since the DROP was executed. You can use the various Flashback Query commands to reverse DML commands, but again, whether this will succeed will depend on what other activity has occurred in the intervening time. It is impossible to roll back a committed transaction, whether DML or DDL. The ACID test does not permit this. The flashback technologies rely on constructing another transaction that will reverse the impact of the original transaction, but it may be that this new transaction will fail, because of other, incompatible committed changes.

Flashback Drop

Accidentally dropping a table is terrifyingly easy to do. It is not just that you can drop the wrong table because of a typing error; it could be the right table, but you are connected to the wrong schema, or logged onto the wrong instance. You can reduce the likelihood of this by setting your SQL*Plus prompt, for example,

```
SQL> set sqlprompt "_user'@'_connect_identifier> "
SYSTEM@ocp10g>
```

TIP To set your sqlprompt automatically for all SQL*Plus sessions, put the preceding command into the glogin.sql file, in the ORACLE_HOME/sqlplus /admin directory.

Flashback Drop lets you reinstate a previously dropped table (but not a truncated table!) exactly as it was before the drop. All the indexes will also be recovered, and also any triggers and grants. Unique, primary key, and not-null constraints will also be recovered, but not foreign key constraints.

EXAM TIP The Flashback Drop command applies only to tables, but all associated objects will also be recovered, except for foreign key constraints.

The Implementation of Flashback Drop

In earlier releases of the Oracle Database, when a table was dropped all references to it were removed from the data dictionary. If it were possible to see the source code for the old DROP TABLE command, you would see that it was actually a series of DELETE commands against the various tables in the SYS schema that define a table and its space usage, followed by a COMMIT. There was no actual clearing of data from disk, but the space used by a dropped table was flagged as being unused and thus available for reuse. Even though the blocks of the table were still there, there was no possible way of getting to them because the data dictionary would have no record of which blocks were part of the dropped table. The only way to recover a dropped table was to do a point-in-time recovery, restoring a version of the database from before the drop when the data dictionary still knew about the table.

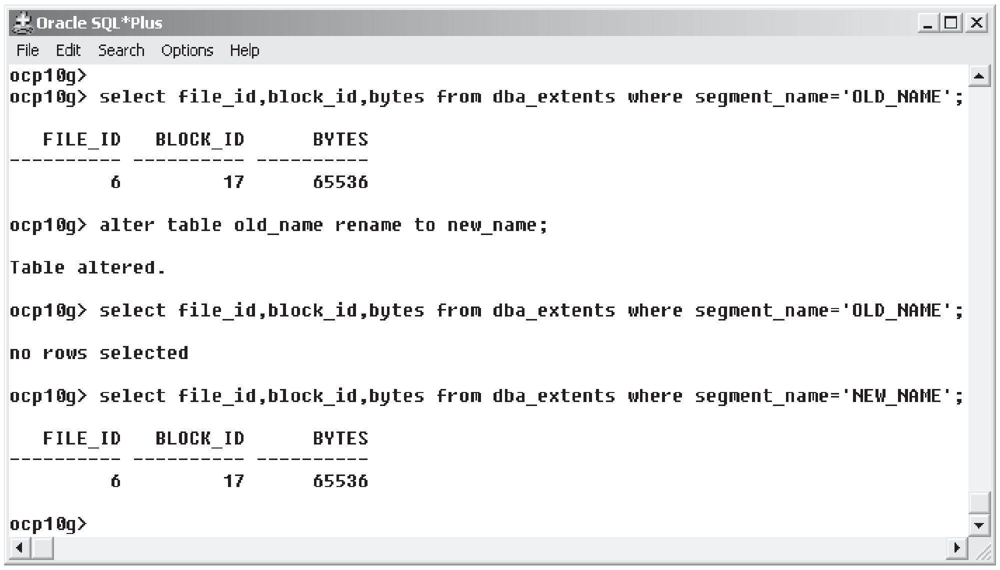
In release 10*g* of the Oracle database, the implementation of the DROP TABLE command has been completely changed. Tables are no longer dropped at all; they are renamed.

In Figure 29-1, you can see that a table, OLD_NAME, occupies one extent of 64KB, which starts in the seventeenth block of file six. After the rename to NEW_NAME, the storage is exactly the same; therefore, the table is the same. Querying the view DBA_OBJECTS would show that the table's object number had not changed either.

The release 10*g* implementation of the DROP TABLE command has been mapped internally onto a RENAME command, which affects the table and all its associated indexes, triggers, and constraints, with the exception of foreign key constraints, which are dropped. Foreign key constraints have to be dropped for real. If they were maintained, even with a different name, then DML on the non-dropped parent table would be constrained by the contents of a dropped table, which would be absurd.

Grants on tables do not have names, so they can't be renamed. But even though when you grant a privilege you specify the object by name, the underlying storage of the grant references the object by number. As the object numbers don't get changed by a RENAME operation, the grants are still valid.

As far as normal SELECT and DML statements are concerned, a dropped table is definitely dropped. There is no change to any other commands, and all your software will assume that a dropped table really is gone. But now that DROP is in fact a RENAME, it becomes possible to un-drop, by renaming the table back to its original name. However, this is not guaranteed to succeed. It may be that the space occupied by the dropped table has been reused. There are also complications if in the interim period another table has been created, reusing the same name as the dropped table.



```
ocp10g>
ocp10g> select file_id,block_id,bytes from dba_extents where segment_name='OLD_NAME';

   FILE_ID   BLOCK_ID      BYTES
---------- ---------- ----------
         6         17      65536

ocp10g> alter table old_name rename to new_name;

Table altered.

ocp10g> select file_id,block_id,bytes from dba_extents where segment_name='OLD_NAME';

no rows selected

ocp10g> select file_id,block_id,bytes from dba_extents where segment_name='NEW_NAME';

   FILE_ID   BLOCK_ID      BYTES
---------- ---------- ----------
         6         17      65536

ocp10g>
```

Figure 29-1 Renaming tables with SQL*Plus

The dropped objects can be queried, by looking at the "recycle bin" to obtain their new names. This is a listing of all objects that have been dropped, mapping the original table and index names onto the system-generated names of the dropped objects. There is a recycle bin for each user, visible in the USER_RECYCLEBIN data dictionary view, or for a global picture, you can query DBA_RECYCLEBIN. The space occupied by the recycle bin objects will be reused automatically when a tablespace comes under space pressure (after which time the objects cannot be recovered), or you can manually force Oracle to really drop the objects with the PURGE command.

TIP There are no guarantees of success with Flashback Drop, but it may well work. The sooner you execute it, the greater the likelihood of success.

Using Flashback Drop

Consider the example in Figure 29-2. This is the most basic use of Flashback Drop. The DROP command renames the table to a system-generated name, and Flashback Drop brings it back. Variations in syntax are

```
SQL> drop table <table_name> purge;
SQL> flashback table <table_name> to before drop rename to <new_name>;
```

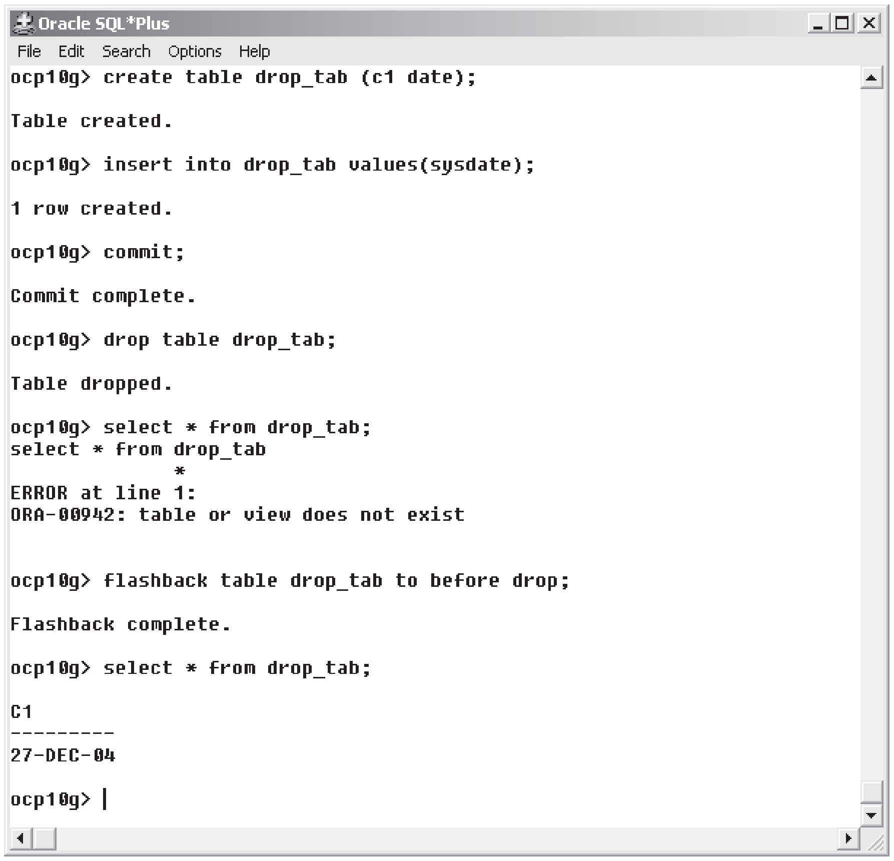
The first command really will drop the table. The PURGE keyword instructs Oracle to revert to the true meaning of DROP: all references to the table are deleted, and it can never be brought back. The second command will flash back the table but give it a new name. This would be essential if, between the drop and the flashback, another table had been created with the same name as the dropped table. Note that although a table can be renamed during a flashback, it cannot change the schema: all flashback operations occur within the schema to which the object belongs. The indexes, triggers, and constraints that are flashed back along with the table keep their recycle bin names. If you want to return them to their original names, you must rename them manually after the flashback.

There are two points to emphasize here. First, Flashback Drop can only recover from a DROP. It cannot recover from a TRUNCATE. Second, if you drop a user with, for example,

```
SQL> drop user scott cascade;
```

you will not be able to recover any of SCOTT's tables with flashback. The drop of the schema means that Oracle cannot maintain the objects at all, even in the recycle bin, because there is no user to connect them to.

Database Control also has an interface to Flashback Drop. From the database home page, take the Maintenance tab and then Perform Recovery in the Backup/Recovery section. In the drop-down box for Object Type, select Tables, and then the



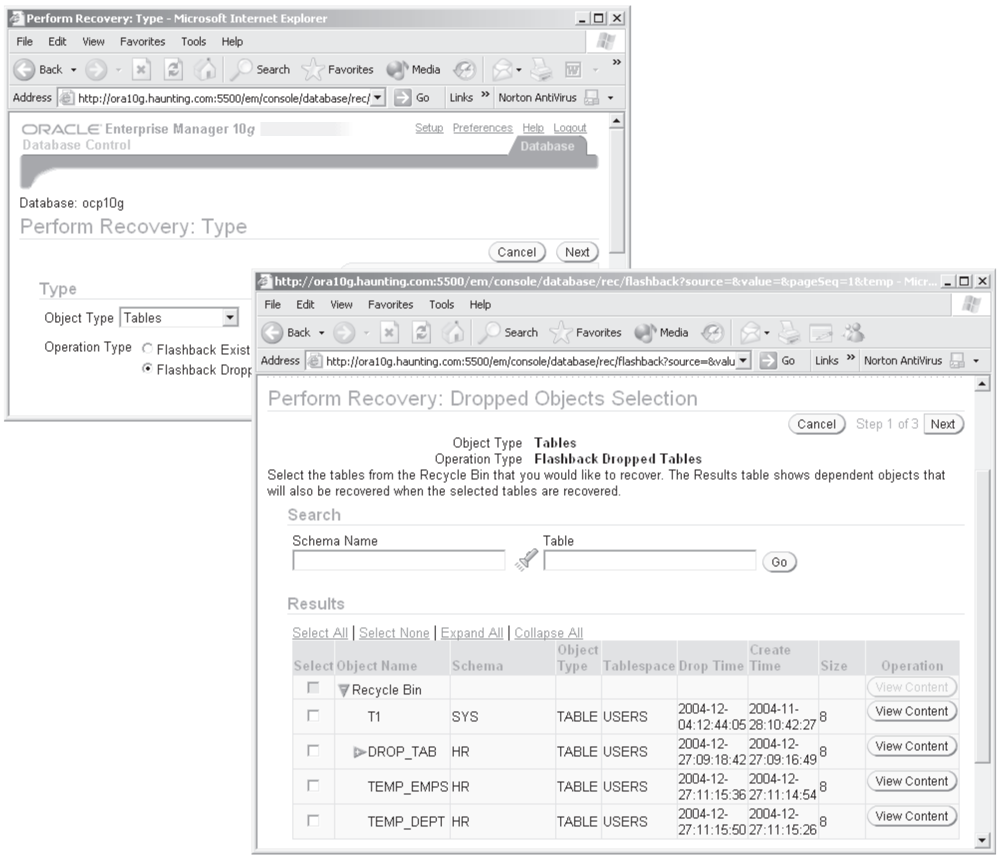
```
ocp10g> create table drop_tab (c1 date);

Table created.

ocp10g> insert into drop_tab values(sysdate);

1 row created.

ocp10g> commit;

Commit complete.

ocp10g> drop table drop_tab;

Table dropped.

ocp10g> select * from drop_tab;
select * from drop_tab
              *
ERROR at line 1:
ORA-00942: table or view does not exist


ocp10g> flashback table drop_tab to before drop;

Flashback complete.

ocp10g> select * from drop_tab;

C1
---------
27-DEC-04

ocp10g> 
```

Figure 29-2 Using Flashback Drop

radio button for Flashback Dropped Tables. This will take you to the Perform Recovery: Dropped Objects Selection window, where you can view all the dropped tables in your database. From here you can select from the dropped tables or go on to recover them, as shown in Figure 29-3.

If a table is dropped and then another table is created with the same name and then also dropped, there will be two tables in the recycle bin. They will have different recycle bin names, but the same original name. By default, a Flashback Drop command will always recover the most recent version of the table, but if this is not the version you want, you can specify the recycle bin name of the version you want recovered, rather than the original name. For example,

```
SQL> flashback table "BIN$sn0WEwXuTum7c1Vx4dOcaA==$0" to before drop;
```


Perform Recovery: Type - Microsoft Internet Explorer
ORACLE Enterprise Manager 10g
Database Control
Setup Preferences Help Logout
Database
Cancel
Next
http://ora10g.haunting.com:5500/em/console/database/rec/flashback?source=&value=&pageSeq=1&temp - Micr...
Cancel
Step 1 of 3
Next
Go
Select All | Select None | Expand All | Collapse All

Database: ocp10g

Perform Recovery: Type

Type

Object Type Tables

Operation Type
( ) Flashback Exist
(●) Flashback Dropp

Perform Recovery: Dropped Objects Selection

Object Type **Tables**
Operation Type **Flashback Dropped Tables**

Select the tables from the Recycle Bin that you would like to recover. The Results table shows dependent objects that will also be recovered when the selected tables are recovered.

Search

Schema Name

Table

Results

Select	Object Name	Schema	Object Type	Tablespace	Drop Time	Create Time	Size	Operation
☐	Recycle Bin							View Content
☐	T1	SYS	TABLE	USERS	2004-12-04:12:44:05	2004-11-28:10:42:27	8	View Content
☐	DROP_TAB	HR	TABLE	USERS	2004-12-27:09:18:42	2004-12-27:09:16:49	8	View Content
☐	TEMP_EMPS	HR	TABLE	USERS	2004-12-27:11:15:36	2004-12-27:11:14:54	8	View Content
☐	TEMP_DEPT	HR	TABLE	USERS	2004-12-27:11:15:50	2004-12-27:11:15:26	8	View Content

Figure 29-3 The Database Control interface to Flashback Drop

Exercise 29-1: Using Flashback Drop with SQL*Plus

Create a new schema, and a table within it. Drop the table, and then recover it with Flashback Drop.

1. Connect to your database as user SYSTEM with SQL*Plus.
2. Create a user for this exercise.

```
SQL> create user dropper identified by dropper;
SQL> grant connect,resource to dropper;
SQL> connect dropper/dropper;
```

3. Create a table, with an index and a constraint, and insert a row.

```
SQL> create table names (name varchar2(10));
SQL> create index name_idx on names(name);
SQL> alter table names add (constraint name_u unique(name));
SQL> insert into names values ('John');
SQL> commit;
```

4. Confirm the contents of your schema.

```
SQL> select object_name,object_type from user_objects;
SQL> select constraint_name,constraint_type,table_name from user_constraints;
```

5. Drop the table.

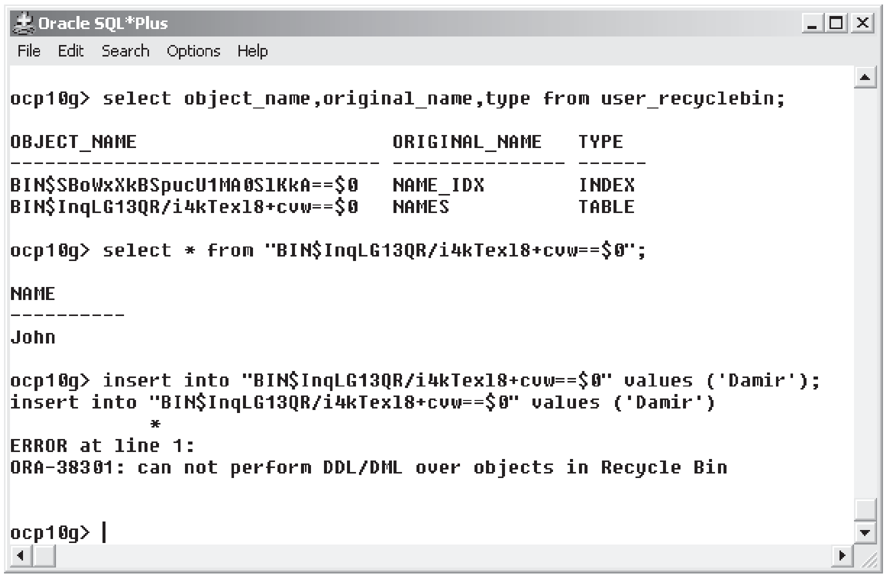
```
SQL> drop table names;
```

6. Re-run the queries from Step 4. Note that the objects and the constraints do still exist but that they now have system-generated names, all prefixed with BIN$.
7. Query your recycle bin to see the mapping of the original name to the recycle bin names.

```
SQL> select object_name,original_name,type from user_recyclebin;
```

8. Demonstrate that it is possible to query the recycle bin but that you cannot do DML against it, as in Figure 29-4. Note that the table name must be enclosed in double quotes to allow SQL*Plus to parse the nonstandard characters correctly.
9. Recover the table with Flashback Drop.

```
SQL> flashback table names to before drop;
```



```
ocp10g> select object_name,original_name,type from user_recyclebin;

OBJECT_NAME                      ORIGINAL_NAME   TYPE
-------------------------------- --------------- ------
BIN$SBoWxXkBSpucU1MA0SlKkA==$0   NAME_IDX        INDEX
BIN$InqLG13QR/i4kTex18+cvw==$0   NAMES           TABLE

ocp10g> select * from "BIN$InqLG13QR/i4kTex18+cvw==$0";

NAME
----------
John

ocp10g> insert into "BIN$InqLG13QR/i4kTex18+cvw==$0" values ('Damir');
insert into "BIN$InqLG13QR/i4kTex18+cvw==$0" values ('Damir')
            *
ERROR at line 1:
ORA-38301: can not perform DDL/DML over objects in Recycle Bin


ocp10g> |
```

Figure 29-4 Querying the recycle bin

10. Re-run the queries from Step 4. Note that the index and the constraint have retained their recycle bin names.
11. Rename the index and constraint to the original names. In these examples, substitute your own recycle bin names:

```
SQL> alter index "BIN$SBoWxXkBSpucU1MA0SlKkA==$0" rename to name_idx;
SQL> alter table names rename constraint "BIN$PfhO1XqPTrewRb5+ToED1Q==$0"
to name_u;
```

12. Confirm the success of the operation by re-running the queries from Step 4.
13. Connect as user SYSTEM, and drop the DROPPER schema.

```
SQL> connect system/oracle;
SQL> drop user dropper cascade;
```

14. Query the DBA_RECYCLEBIN view to demonstrate that all the objects owned by user DROPPER really are gone.

```
SQL> select count(*) from dba_recyclebin where owner='DROPPER';
```

Managing the Recycle Bin

The recycle bin is a term given to the storage space used by dropped objects. You can ignore the recycle bin completely—its management is automatic, both in terms of transferring objects into it when they are dropped, and removing them permanently when the space is needed in the tablespace for live objects. But there may be circumstances when you will need to be aware of the contents of the recycle bin and how much space they are taking up.

TIP The recycle bin can be disabled with a hidden parameter.

Querying the Recycle Bin

Each user has his own recycle bin: s/he can always view dropped tables in his or her own schema. The simplest way is the SHOW RECYCLEBIN command:

```
SQL> show recyclebin;
ORIGINAL NAME RECYCLEBIN NAME                OBJECT TYPE  DROP TIME
------------- ------------------------------ ------------ -----------
DROP_TAB      BIN$vWMhmt3sTcqJ9WhSREM29g==$0 TABLE        2004-12-27:09:18:42
TEMP_DEPT     BIN$OLp3r9zPRRe6KSjs7Ee3gQ==$0 TABLE        2004-12-27:11:15:50
TEMP_EMPS     BIN$DKaQ10DDSty8hXQH2Xniwg==$0 TABLE        2004-12-27:11:15:36
```

This shows that the current user has three dropped tables: their original names, their recycle bin names, and the time they were dropped. For more detailed information, query the data dictionary view USER_RECYCLEBIN, or DBA_RECYCLEBIN for a global view:

```
SQL> select owner,original_name,type,droptime,can_undrop, space from
dba_recyclebin;
```

```
OWNER      ORIGINAL_NAME   TYPE   DROPTIME            CAN   SPACE
---------- --------------- ------ ------------------- --- -------
SYS        T1              TABLE  2004-12-04:12:44:05 YES       8
DROPPER    T1              TABLE  2004-12-27:11:23:21 YES       8
HR         DROP_TAB        TABLE  2004-12-27:09:18:42 YES       8
HR         TEMP_EMPS       TABLE  2004-12-27:11:15:36 YES       8
HR         TEMP_DEPT       TABLE  2004-12-27:11:15:50 YES       8
```

The critical column is CAN_UNDROP. Oracle is under no obligation to keep dropped tables or indexes: the Flashback Drop facility is purely a convenience that Oracle provides; it is not part of the relational database standard. If Oracle needs the space being occupied by a dropped object to allocate more space to a live object, it will take it; from that point, the dropped object can no longer be recovered with Flashback Drop, and it will be removed from the view. The SPACE column (in units of datafile blocks) shows how much space is taken up by the dropped object.

Having identified the dropped table's name in the recycle bin, it can be queried like any other table, though you will have to enclose its name in double quotes because of the nonstandard characters used in recycle bin names. But always remember that you have a limited (and unpredictable) time during which you can do this. If you think it is likely that a dropped table will be needed, you should recover it immediately.

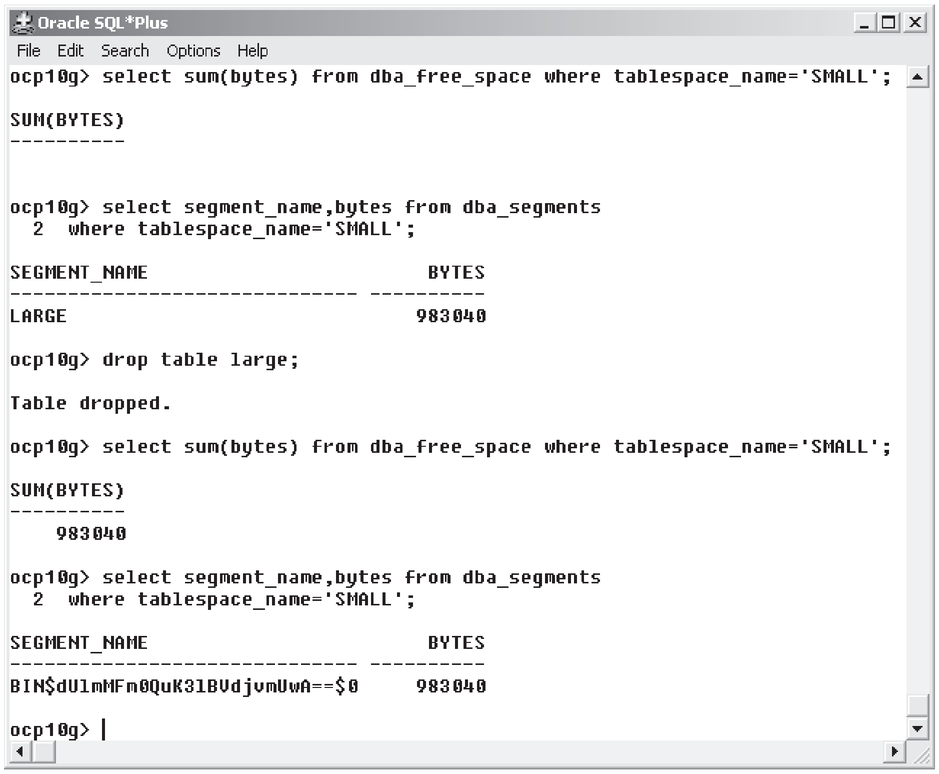
EXAM TIP Flashback Drop is not enabled for tables stored in the SYSTEM tablespace: such tables will not be reported by the queries described in the preceding text, because they are dropped and purged immediately.

Reclaiming Space from the Recycle Bin

Space taken up by dropped objects is in an ambiguous state: it is assigned to the object, but Oracle can overwrite it at will. The normal diagnostics regarding space usage will ignore space occupied by the recycle bin. This means that your "tablespace percent full" alerts will not fire until the warning and critical space usage levels are reached by live objects. Furthermore, if your datafiles have the AUTOEXTEND attribute enabled, Oracle will not in fact autoextend the datafiles until all space occupied by dropped objects has been reassigned: it will overwrite the recycle bin in preference to increasing the datafile size.

Consider this example: a 1MB tablespace, called SMALL, has been completely filled by one table, called LARGE. The space usage alerts will have fired, and querying DBA_FREE_SPACE reports no space available. Then the table is dropped. The alert will clear itself and DBA_FREE_SPACE will report that the whole tablespace is empty, but querying the recycle bin, or indeed DBA_SEGMENTS, will tell the truth, as in Figure 29-5.

This apparently contradictory state is resolved by Oracle reusing space as it needs it. If space is required in the tablespace for a new segment, then it will be taken, and it will no longer be possible to retain the dropped table. If there are many deleted objects in the recycle bin, Oracle will overwrite the object that had been in there for



```
ocp10g> select sum(bytes) from dba_free_space where tablespace_name='SMALL';

SUM(BYTES)
----------


ocp10g> select segment_name,bytes from dba_segments
  2  where tablespace_name='SMALL';

SEGMENT_NAME                        BYTES
------------------------------ ----------
LARGE                              983040

ocp10g> drop table large;

Table dropped.

ocp10g> select sum(bytes) from dba_free_space where tablespace_name='SMALL';

SUM(BYTES)
----------
    983040

ocp10g> select segment_name,bytes from dba_segments
  2  where tablespace_name='SMALL';

SEGMENT_NAME                        BYTES
------------------------------ ----------
BIN$dUlmMFm0QuK31BVdjvmUwA==$0     983040

ocp10g> |
```

Figure 29-5 Recycle bin space usage

the longest time. This FIFO, or first in, first out, algorithm assumes that objects dropped recently are most likely candidates for a flashback.

It is also possible to remove deleted objects permanently using the PURGE command in its various forms:

```
drop table <table_name> purge
```

means drop the table and do not move it to the recycle bin, and

```
purge table <table_name>
```

means remove the table from the recycle bin. If there are several objects with the same original name, the oldest is removed. Avoid this confusion by specifying the recycle bin name instead. Use

```
purge index <index_name>
```

to remove an index from the recycle bin; again, you can specify either the original name, or the recycle bin name. The form

```
purge tablespace <tablespace_name>
```

means remove all dropped objects from the tablespace; the form

```
purge tablespace <tablespace_name> user <user_name>
```

means remove all dropped objects belonging to one user from the tablespace; the form

```
purge user_recyclebin
```

means remove all your dropped objects; and the form

```
purge dba_recyclebin
```

means remove all dropped objects. You will need DBA privileges to execute this.

Flashback Query

The basic form of Flashback Query has been available since release 9*i* of the Oracle Database: you can query the database as it was at some time in the past. The principle is that your query specifies a time that is mapped onto a system change number, an SCN, and that whenever the query hits a block that has been changed since that SCN, it will go to the undo segments to extract the undo data needed to roll back the change. This rollback is strictly temporary and only visible to the session running the Flashback Query. Clearly, for a Flashback Query to succeed, the undo data must be available.

In release 10*g* of the database, Flashback Query has been enhanced substantially, and it can now be used to retrieve all versions of a row, to reverse individual transactions, or to reverse all the changes made to a table since a certain time. It is also possible to guarantee that a flashback will succeed, but there is a price to be paid for enabling this: it may cause transactions to fail.

EXAM TIP All forms of Flashback Query rely on undo data to reconstruct data as it was at an earlier point in time.

Basic Flashback Query

Any one select statement can be directed against a previous version of a table. Consider this example:

```
ocp10g> select sysdate from dual;
SYSDATE
-----------------
27-12-04 16:54:06
ocp10g> delete from regions where region_name like 'A%';
```

```
2 rows deleted.
ocp10g> commit;
Commit complete.
ocp10g> select * from regions;
REGION_ID REGION_NAME
--------- -------------------------
        1 Europe
        4 Middle East and Africa
ocp10g> select * from regions as of timestamp to_timestamp('27-12-04
16:54:06','dd-mm-yy hh24:mi:ss');
REGION_ID REGION_NAME
--------- -------------------------
        1 Europe
        2 Americas
        3 Asia
        4 Middle East and Africa
ocp10g> select * from regions as of timestamp to_timestamp('27-12-04
16:54:06','dd-mm-yy hh24:mi:ss') minus select * from regions;
REGION_ID REGION_NAME
--------- -------------------------
        2 Americas
        3 Asia
```

First, note the time. Then delete some rows from a table and commit the change. A query confirms that there are only two rows in the table, and no rows where the REGION_NAME begins with *A*. The next query is directed against the table as it was at the earlier time: back then there were four rows, including those for "Asia" and "Americas." Make no mistake about this: the two rows beginning with *A* are gone; they were deleted, and the delete was committed. It cannot be rolled back. The deleted rows you are seeing have been constructed from undo data. The final query combines real-time data with historical data, to see what rows have been removed. The output of this query could be used for repair purposes, to insert the rows back into the table.

While being able to direct one query against data as of an earlier point in time may be useful, there will be times when you want to make a series of selects. It is possible to take your whole session back in time by using the DBMS_FLASHBACK package:

```
ocp10g> execute dbms_flashback.enable_at_time(-
> to_timestamp('27-12-04 16:54:06','dd-mm-yy hh24:mi:ss'));
PL/SQL procedure successfully completed.
ocp10g>
```

From this point on, all queries will see the database as it was at the time specified. All other sessions will see real-time data, but this one session will see a frozen version of the database, until the flashback is cancelled:

```
ocp10g> execute dbms_flashback.disable;
PL/SQL procedure successfully completed.
ocp10g>
```

While in flashback mode, it is impossible to execute DML commands. They will throw an error. Only SELECT is possible.

How far back you can take a Flashback Query (either one query, or by using DBMS_FLASHBACK) is dependent on the contents of the undo segments. If the undo data needed to construct the out-of-date result set is not available, then the query will fail with an "ORA-08180: no snapshot found based on specified time" error.

The syntax for enabling Flashback Query will accept either a timestamp or an SCN. If you use an SCN, then the point to which the flashback goes is precise. If you specify a time, it will be mapped onto an SCN with a precision of three seconds.

EXAM TIP You can query the database as of an earlier point-in-time, but you can never execute DML against the older versions of the data.

Flashback Table Query

Conceptually, a table flashback is simple. Oracle will query the undo segments to extract details of all rows that have been changed, and then it will construct and execute statements that will reverse the changes. The flashback operation is a separate transaction that will counteract the effect of all the previous transactions, if possible. The database remains online and normal work is not affected, unless row locking is an issue. This is not a rollback of committed work; it is a new transaction designed to reverse the effects of committed work. All indexes are maintained, and constraints enforced: a table flashback is just another transaction, and the usual rules apply. The only exception to normal processing is that by default triggers on the table are disabled for the flashback operation.

A table flashback will often involve a table that is in a foreign key relationship. In that case, it is almost inevitable that the flashback operation will fail with a constraint violation. To avoid this problem, the syntax permits flashback of multiple tables with one command, which will be executed as a single transaction with the constraint checked at the end.

The first step to enabling table flashback is to enable row movement on the tables. This is a flag set in the data dictionary that informs Oracle that rowids may change. A rowid can never actually change, but a flashback operation may make it appear as though it has. For instance, in the case of a row that is deleted, the flashback operation will insert it back into the table: it will have the same primary key value, but a different rowid.

In the example that follows, there are two tables: EMP and DEPT. There is a foreign key relationship between them, stating that every employee in EMP must be a member of a department in DEPT.

First, insert a new department and an employee in that department, and note the time:

```
ocp10g> insert into dept values(50,'SUPPORT','LONDON');
1 row created.
ocp10g> insert into emp values(8000,'WATSON','ANALYST',7566,'27-DEC-
04',3000,null,50);
1 row created.
```

```
ocp10g> commit;
Commit complete.
ocp10g> select sysdate from dual;
SYSDATE
-----------------
27-12-04 18:30:11
```

Next delete the department and the employee, taking care to delete the employee first to avoid a constraint violation:

```
ocp10g> delete from emp where empno=8000;
1 row deleted.
ocp10g> delete from dept where deptno=50;
1 row deleted.
ocp10g> commit;
Commit complete.
```

Now attempt to flash back the tables to the time when the department and employee existed:

```
ocp10g> flashback table emp to timestamp to_timestamp('27-12-04
18:30:11','dd-mm-yy hh24:mi:ss');
flashback table emp to timestamp to_timestamp('27-12-04 18:30:11','dd-mm-yy
hh24:mi:ss')
                *
ERROR at line 1:
ORA-08189: cannot flashback the table because row movement is not enabled
```

This fails because by default row movement, which is a prerequisite for table flashback, is not enabled for any table, so enable it, for both tables:

```
ocp10g> alter table dept enable row movement;
Table altered.
ocp10g> alter table emp enable row movement;
Table altered.
```

and now try the flashback again:

```
ocp10g> flashback table emp to timestamp to_timestamp('27-12-04
18:30:11','dd-mm-yy hh24:mi:ss');
flashback table emp to timestamp to_timestamp('27-12-04 18:30:11','dd-mm-yy
hh24:mi:ss')
*
ERROR at line 1:
ORA-02091: transaction rolled back
ORA-02291: integrity constraint (SCOTT.FK_DEPTNO) violated - parent key not
found
```

This time the flashback fails for a more subtle reason. The flashback is attempting to reverse the deletion of employee 8000 by inserting him, but employee 8000 was in department 50, which has been deleted and so does not exist. Hence the foreign key violation. You could avoid this problem by flashing back the DEPT table first, which would insert department 50. But if your flashback involves many tables and many

DML statements, it may be logically impossible to find a sequence that will work. The answer is to flash back both tables together:

```
ocp10g> flashback table emp,dept to timestamp to_timestamp('27-12-04
18:30:11','dd-mm-yy hh24:mi:ss');
Flashback complete.
```

This succeeds because both the tables are flashed back in one transaction, and the constraints are only checked at the end of that transaction, by which time, the data is logically consistent.

The flashback could still fail for other reasons:

- Primary key violations will occur if a key value has been reused between a delete and the flashback.
- An "ORA-08180: no snapshot found based on specified time" will be raised if there is not enough undo information to go back to the time requested.
- If any rows affected by the flashback are locked by other users, the flashback will fail with "ORA-00054: resource busy and acquire with NOWAIT specified."
- The table definitions must not change during the period concerned; flashback cannot go across DDLs. Attempting to do this will generate "ORA-01466: Unable to read data – table definition has changed."
- Flashback does not work for tables in the SYS schema. Try to imagine the effect of flashing back part of the data dictionary....

If a table flashback fails for any reason, the flashback operation will be cancelled: any parts of it that did succeed will be rolled back, and the tables will be as they were before the flashback command was issued.

Variations in the syntax allow flashback to a system change number, and firing of DML triggers during the operation:

```
SQL> flashback table emp,dept to scn 6539425 enable triggers;
```

Table flashback can also be initiated by Database Control. From the database home page, take the Maintenance tab and then the Perform Recovery link in the Backup/Recovery section. Select Tables in the Object Type drop-down box, and then the Flashback Existing Tables radio button to invoke the Flashback Table Wizard. However table flashback is initiated, it is always recorded in the alert log.

Flashback Versions Query

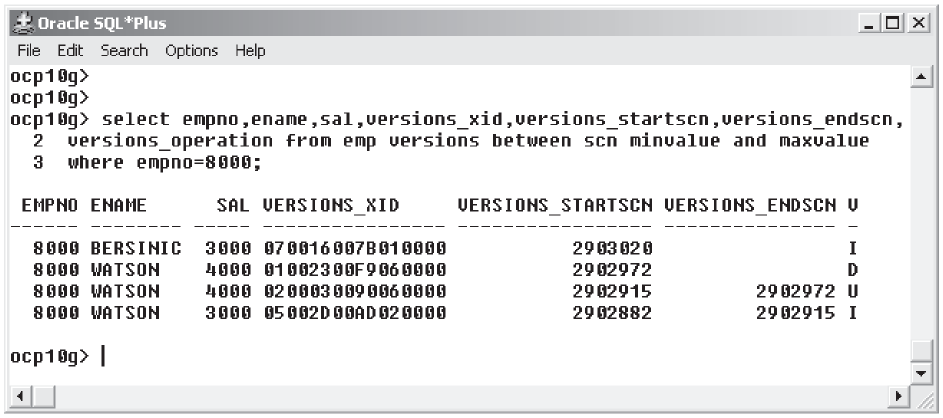
A row may have changed several times during its life. Flashback Versions Query lets you see all the committed versions of a row (but not any uncommitted versions), including the timestamps for when each version was created and when it ended. You can also see the transaction identifier of the transaction that created any given version of a row, which can then be used with Flashback Transaction Query. This information is exposed by a number of pseudo-columns that are available with every table.

Pseudo-columns are columns appended to the row by Oracle internally; they are not part of the ISO standards for a relational database, but they can be very useful. One pseudo-column is the row ID: the unique identifier for every row in the database, that is used in indexes as the pointer back to the table. These pseudo-columns are relevant to flashback:

- **VERSIONS_STARTSCN** The SCN at which this version of the row was created, either by INSERT or by UPDATE
- **VERSIONS_STARTTIME** The timestamp at which this version of the row was created
- **VERSIONS_ENDSCN** The SCN at which this version of the row expired, either because of DELETE or UPDATE
- **VERSIONS_ENDTIME** The timestamp at which this version of the row expired
- **VERSIONS_XID** The unique identifier for the transaction that created this version of the row
- **VERSIONS_OPERATIONS** The operation done by the transaction to create this version of the row, either INSERT or UPDATE or DELETE

To see these pseudo-columns, you must include the VERSIONS BETWEEN keywords in your query. For example, Figure 29-6 shows all versions of the row for employee 8000.

The versions are sorted in descending order of existence: they must be read from the bottom up. The bottom row shows that employee 8000 was inserted (the *I* in the last column) at SCN 2902882 by transaction number 05002D00AD020000. The employee was given an ENAME of WATSON and a SAL of 3000. This version of the row existed until SCN 2902915, which takes us to the third row. At this SCN, the row



```
ocp10g>
ocp10g>
ocp10g> select empno,ename,sal,versions_xid,versions_startscn,versions_endscn,
  2  versions_operation from emp versions between scn minvalue and maxvalue
  3  where empno=8000;

 EMPNO ENAME      SAL VERSIONS_XID     VERSIONS_STARTSCN VERSIONS_ENDSCN V
------ -------- ----- ---------------- ----------------- --------------- -
  8000 BERSINIC  3000 070016007B010000           2903020                 I
  8000 WATSON    4000 01002300F9060000           2902972                 D
  8000 WATSON    4000 0200030090060000           2902915         2902972 U
  8000 WATSON    3000 05002D00AD020000           2902882         2902915 I

ocp10g> |
```

Figure 29-6 Flashback Versions Query

was updated (the *U* in the last column) with a new salary. This version of row persisted until SCN 2902972, when it was deleted, as shown in the second row. The VERSIONS_ENDSCN column is always null for a deletion. The top row of the result set shows a new insertion, which reuses the employee number. For this row, the VERSIONS_ENDSCN is also null, because the row still exists, in that version, as at the end of the time range specified in the query.

In the example in Figure 29-6, the VERSIONS BETWEEN uses two constants for the SCN. MINVALUE instructs Oracle to retrieve the earliest information in the undo segments; MAXVALUE will be the current SCN. In other words, the query as written will show all versions that can possibly be retrieved, given the information available. The syntax will also accept a range specified with two timestamps:

```
SQL> select empno,ename,sal,versions_xid,versions_starttime,
versions_endtime,versions_operation from emp versions between
timestamp to_timestamp(systimestamp - 1/24) and systimestamp
where empno=8000;
```

This example will select all versions of employee number 8000 that existed during the last hour.

Flashback Transaction Query

Flashback Table Query and Flashback Versions Query use undo data for an object. Flashback Transaction Query analyzes the undo by a different dimension: it will retrieve all the undo data for a transaction, no matter how many objects it affects. The critical view is FLASHBACK_TRANSACATION_QUERY, described here:

```
ocp10g> describe flashback_transaction_query
 Name                                   Null?    Type
 ----------------------------------- -------- ---------------------
 XID                                             RAW(8)
 START_SCN                                       NUMBER
 START_TIMESTAMP                                 DATE
 COMMIT_SCN                                      NUMBER
 COMMIT_TIMESTAMP                                DATE
 LOGON_USER                                      VARCHAR2(30)
 UNDO_CHANGE#                                    NUMBER
 OPERATION                                       VARCHAR2(32)
 TABLE_NAME                                      VARCHAR2(256)
 TABLE_OWNER                                     VARCHAR2(32)
 ROW_ID                                          VARCHAR2(19)
 UNDO_SQL                                        VARCHAR2(4000)
```

Because the data in this view may be sensitive, it is protected by a privilege: you must have been granted SELECT ANY TRANSACTION before you can query it. By default, this privilege is granted to SYS and to the DBA role. There will be one or more rows in this view for every transaction whose undo data still exists in the undo segments, and every row will refer to one row affected by the transaction. Table 29-1 describes the columns.

Column Name	Description
XID	The transaction identifier. This is the join column to the pseudo-column VERSIONS_XID displayed in a Flashback Versions Query
START_SCN	The system change number at the time the transaction started
START_TIMESTAMP	The timestamp at the time the transaction started
COMMIT_SCN	The system change number at the time the transaction was committed
COMMIT_TIMESTAMP	The timestamp at the time the transaction was committed
LOGON_USER	The Oracle username of the session that performed the transaction
UNDO_CHANGE#	The undo system change number. This is not likely to be relevant to most work
OPERATION	The DML operation applied to the row: INSERT, UPDATE, or DELETE
TABLE_NAME	The table to which the row belongs
TABLE_OWNER	The schema to which the table belongs
ROW_ID	The unique identifier of the row affected
UNDO_SQL	A constructed statement that will reverse the operation. For example, if the OPERATION were a DELETE, then this will be an INSERT

Table 29-1 Flashback Transaction Query Columns

A one-line SQL statement might generate many rows in FLASHBACK_TRANSACTION_QUERY. This is because SQL is a set-oriented language: one statement can affect many rows. But each row affected will have its own row in the view. The view will show committed transactions and also transactions in progress. For an active transaction, the COMMIT_SCN and COMMIT_TIMESTAMP columns are NULL. Rolled-back transactions are not displayed.

Take an example where a salary was multiplied by eleven, rather than being incremented by ten percent:

```
SQL> update emp set sal =sal*11 where empno=7902;
1 row updated.
SQL> commit;
Commit complete.
```

Later, it is suspected that there was a mistake made. So query the versions of the row:

```
SQL> select ename,sal,versions_xid from emp versions between scn
  2  minvalue and maxvalue where empno=7902;
ENAME             SAL VERSIONS_XID
---------- ---------- ----------------
FORD            33000 06002600B0010000
FORD             3000
```

This does indicate what happened, and it gives enough information to reverse the change. But what if the transaction affected other rows in other tables? To be certain, query FLASHBACK_TRANSACTION_QUERY, which will have one row for every row affected by the transaction. A minor complication is that the XID column is type RAW, whereas the VERSIONS_XID pseudo-column is hexadecimal, so you must use a type-casting function to make the join:

```
SQL> select operation,undo_sql from flashback_transaction_query
where xid=hextoraw('06002600B0010000');
OPERATION  UNDO_SQL
---------- -----------------------------------------------------
UPDATE     update "SCOTT"."EMP" set "SAL" = '3000' where ROWID =
           'AAAM+yAAEAAAAAeAAM';
```

This query returns only one row, which confirms that there was indeed only one row affected by the transaction, and provides a statement that will reverse the impact of the change. Note the use of a ROWID in the UNDO_SQL statement. Provided that there has been no reorganization of the table, this will guarantee that the correct row is changed.

Exercise 29-2: Using Flashback Query with Database Control

Create a table, execute some DML, and then investigate and reverse the changes.

1. Connect to your database as user SYSTEM using SQL*Plus.
2. Create a table and enable row movement for it.

   ```
   SQL> create table countries (name varchar2(10));
   SQL> alter table countries enable row movement;
   ```

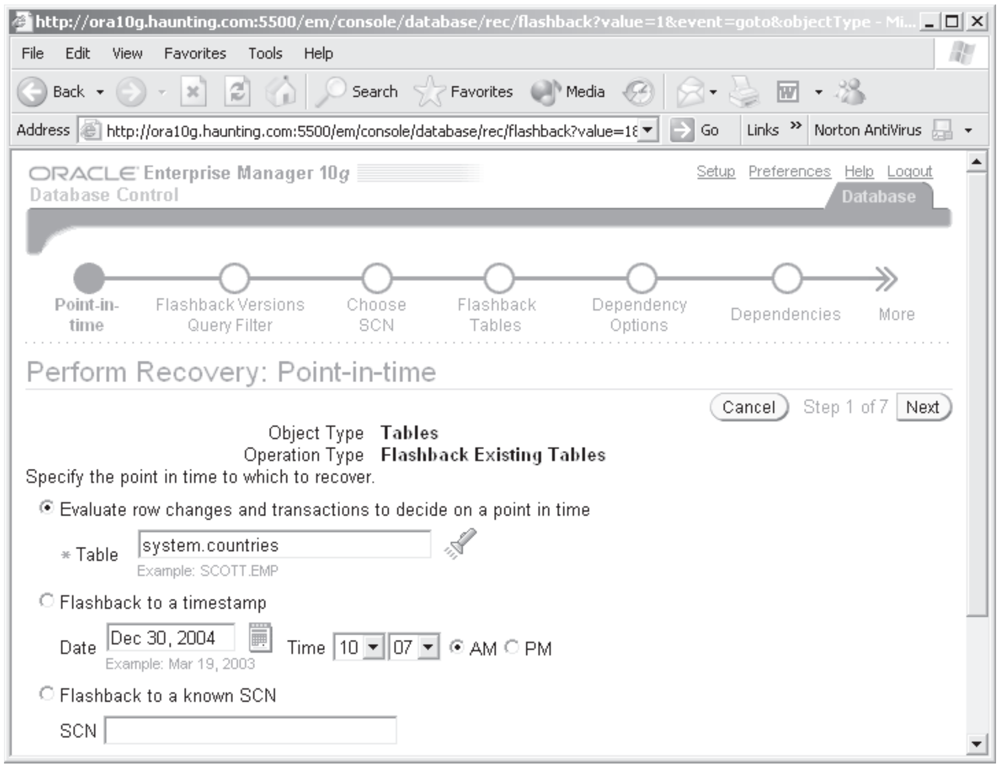
3. Insert some rows as follows.

   ```
   SQL> insert into countries values('Zambia');
   SQL> insert into countries values('Zimbabwe');
   SQL> insert into countries values ('Zamibia');
   SQL> commit;
   ```

   Correct the spelling mistake, but omit the WHERE clause.

   ```
   SQL> update countries set name= 'Namibia';
   SQL> commit;
   ```

4. Connect to your database as user SYSTEM with Database Control.
5. From the database home page, take the Maintenance tab and then the Perform Recovery link in the Backup/Recovery section to reach the Perform Recovery: Type window.
6. In the Object Type drop-down box, choose Tables, and select the Flashback Existing Tables radio button. Click Next to reach the Perform Recovery: Point-in-time window.
7. Select the “Evaluate row changes and transactions to decide on point in time” radio button, and enter **SYSTEM.COUNTRIES** as the table, as in Figure 29-7. Click Next to reach the Perform Recovery: Flashback Versions Query Filter window.


http://ora10g.haunting.com:5500/em/console/database/rec/flashback?value=1&event=goto&objectType - Mi...
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://ora10g.haunting.com:5500/em/console/database/rec/flashback?value=1 Go Links Norton AntiVirus
ORACLE Enterprise Manager 10g Database Control
Setup Preferences Help Logout
Database
Point-in-time
Flashback Versions Query Filter
Choose SCN
Flashback Tables
Dependency Options
Dependencies
More

Perform Recovery: Point-in-time

Cancel Step 1 of 7 Next

Object Type **Tables**
Operation Type **Flashback Existing Tables**

Specify the point in time to which to recover.

[✓ Evaluate row changes and transactions to decide on a point in time]

* Table system.countries
Example: SCOTT.EMP

Flashback to a timestamp

Date Dec 30, 2004 Time 10 07 [✓ AM] PM
Example: Mar 19, 2003

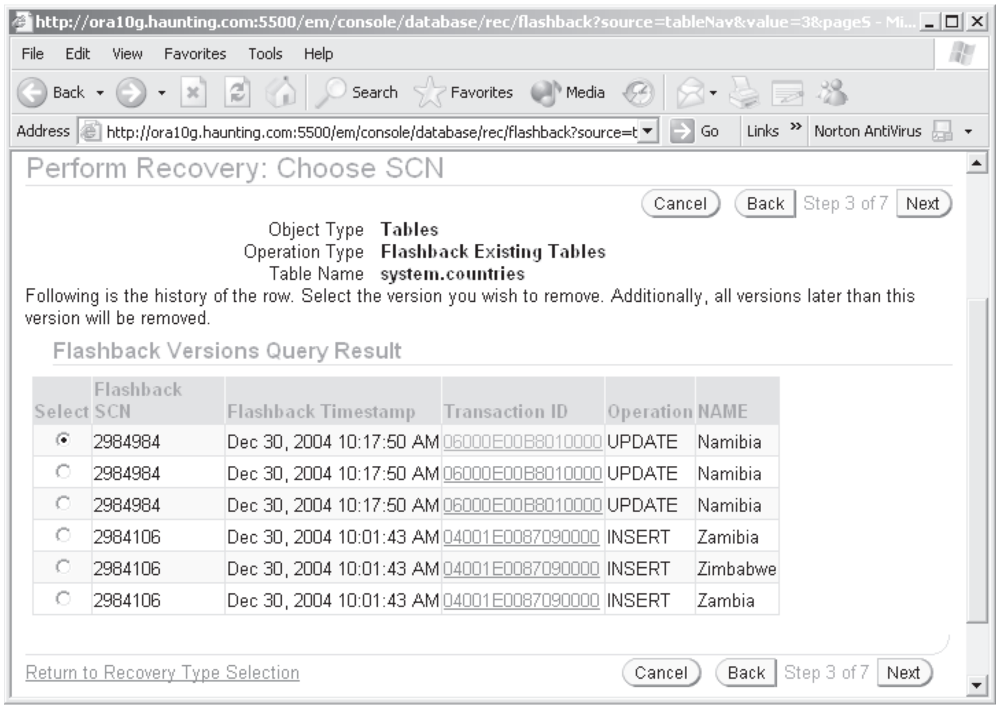
Flashback to a known SCN

SCN

Figure 29-7 Select a table for Flashback Query.

8. In the "Step 1" section, highlight the column NAME and click the Move link to select it for the query. In the "Step 2" section, enter **WHERE NAME LIKE '%'** in order to see all rows. In the "Step 3" section, select the Show All Row History radio button.

   Click Next to reach the Perform Recovery: Choose SCN window shown in Figure 29-8.
9. Note that there are two transactions: the first did the three inserts; the second updated the rows. Select the radio button for one of the "Namibia" rows, and click Next to reach the Perform Recovery: Flashback Tables window.
10. Click Next to reach the Perform Recovery: Review window. Click Show Row Changes to see what will be done by this operation, and Show SQL to see the actual FLASHBACK TABLE statement. Click Submit to execute it.
11. In your SQL*Plus session, confirm that the rows are now back as when first inserted, complete with the spelling mistake.
```
SQL:> select * from countries;
```
12. Tidy up.
```
SQL> drop table countries;
```


http://ora10g.haunting.com:5500/em/console/database/rec/flashback?source=tableNav&value=3&pageS - Mi...
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://ora10g.haunting.com:5500/em/console/database/rec/flashback?source=t Go Links Norton AntiVirus
Cancel Back Step 3 of 7 Next
Return to Recovery Type Selection

Perform Recovery: Choose SCN

Object Type **Tables**
Operation Type **Flashback Existing Tables**
Table Name **system.countries**

Following is the history of the row. Select the version you wish to remove. Additionally, all versions later than this version will be removed.

Flashback Versions Query Result

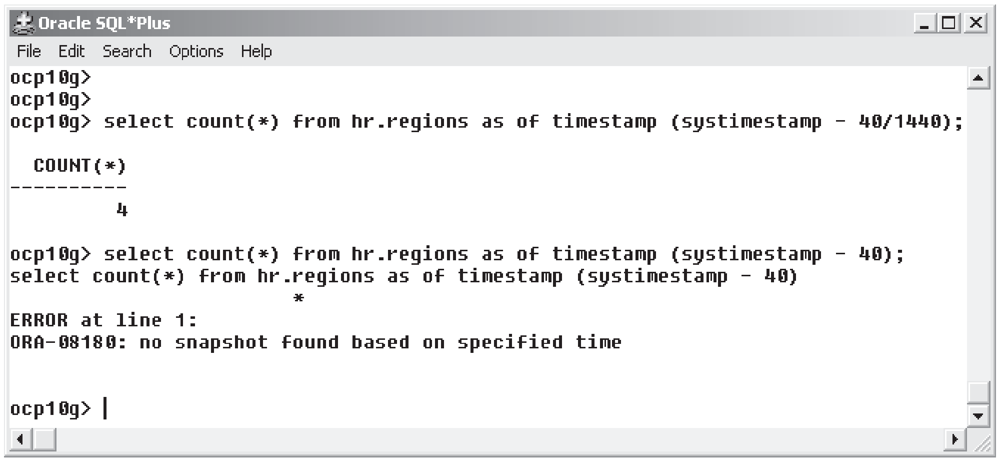
Select	Flashback SCN	Flashback Timestamp	Transaction ID	Operation	NAME
●	2984984	Dec 30, 2004 10:17:50 AM	06000E00B8010000	UPDATE	Namibia
○	2984984	Dec 30, 2004 10:17:50 AM	06000E00B8010000	UPDATE	Namibia
○	2984984	Dec 30, 2004 10:17:50 AM	06000E00B8010000	UPDATE	Namibia
○	2984106	Dec 30, 2004 10:01:43 AM	04001E0087090000	INSERT	Zamibia
○	2984106	Dec 30, 2004 10:01:43 AM	04001E0087090000	INSERT	Zimbabwe
○	2984106	Dec 30, 2004 10:01:43 AM	04001E0087090000	INSERT	Zambia

Figure 29-8 Flashback Versions Query

Flashback and Undo Data

Flashback query in its various forms relies entirely on undo data. You are asking Oracle to present you a version of the data as it was some time ago; if the data has been changed since that time, Oracle must roll back the changes. To do this, Oracle needs the undo data that protected the change. Whether the query will succeed will depend on whether that undo data is still available. Consider Figure 29-9. The first query asks for a view of the table as it was forty minutes ago, and it succeeds. This is because there is at least forty minutes of undo data available in the undo segments. The second query attempts to go back forty days, and it fails. In virtually all databases, it would be completely unrealistic to expect Flashback Query to work over such a long period. You would need an undo tablespace the size of Jupiter to store that much undo data.

Undo data is generated as necessary according to the transaction workload: at busy times of day you may be generating many megabytes of undo per second; at other times you may be generating virtually no undo at all. As undo data can be overwritten once the transaction is completed, the age of the oldest undo data available in the undo tablespace will vary depending on workload. A comprehensive discussion



```
ocp10g>
ocp10g>
ocp10g> select count(*) from hr.regions as of timestamp (systimestamp - 40/1440);

  COUNT(*)
----------
         4

ocp10g> select count(*) from hr.regions as of timestamp (systimestamp - 40);
select count(*) from hr.regions as of timestamp (systimestamp - 40)
                        *
ERROR at line 1:
ORA-08180: no snapshot found based on specified time


ocp10g> |
```

Figure 29-9 Flashback Query and undo data

of this was included in Chapter 16 with reference to the "ORA-1555: snapshot too old" error, and this is equally relevant to Flashback Query.

To guarantee that a Flashback Query will always succeed for a given period, set the RETENTION GUARANTEE attribute for the undo tablespace, in conjunction with the UNDO_RETENTION instance parameter. This will ensure that you can always flash back the number of seconds specified, but the price you will pay is that if your undo tablespace is not sized adequately for the transaction workload, then the database may hang for DML. As discussed in Chapter 16, you must monitor the V$UNDOSTAT view to calculate the necessary size.

Chapter Review

The Flashback technologies detailed in this chapter have been Flashback Drop and the various options for Flashback Query. These are completely different from the Flashback Database described in Chapter 28.

Flashback Drop lets you "undrop" a table, together with any associated objects. It is actually implemented by renaming dropped objects rather than physically dropping them, which puts them in the recycle bin. They are only actually dropped when the space they occupy is needed for live objects, or by the use of the PURGE command. There is no guarantee that a Flashback Drop will succeed; this will depend on whether the space used by the dropped object has been reused.

Flashback Query allows you to query the database as it was at some time in the past; it is implemented by the use of undo data. The simplest version of Flashback Query is to use the AS OF keyword in a SELECT statement. Flashback Versions Query displays all versions of a row between two times, with various pseudo-columns to identify what happened. Flashback Table Query uses the Flashback Versions Query data to reverse changes made to a table; Flashback Transaction Query uses it to reverse changes made by a transaction. By default there is no guarantee that a Flashback Query will succeed, but by setting the RETENTION GUARANTEE attribute for your undo tablespace, you can change this.

These Flashback technologies are online operations; your end users will not notice that they are in progress. They are enabled automatically and are always available when the applicable privileges have been granted.

Questions

1. Different Flashback technologies use different data structures. For each technology, A to G, choose the data structure, a to e, upon which it relies.

A. Use of DBMS_FLASHBACK	a. The redo log (online and/or archive)
B. Flashback Database	b. Undo segments
C. Flashback Drop	c. The recycle bin
D. Flashback Table Query	d. The flashback log
E. Flashback Transaction Query	e. RMAN backups, in the flash recovery area
F. Flashback Version Query	
G. SELECT … AS OF ….	

2. When you drop a table, what objects will go into the recycle bin? (Select all that apply.)
   - **A.** The table
   - **B.** Grants on the table
   - **C.** Indexes on the table
   - **D.** All constraints on the table
   - **E.** All constraints on the table except foreign key constraints
3. After dropping a table, how can you access the rows within it? (Choose the best answer.)
   - **A.** Query the table using the AS OF syntax.
   - **B.** Query the table using the BEFORE DROP syntax.
   - **C.** Query the table using its recycle bin name.
   - **D.** You can't query it until it has been recovered.
4. If a table has been dropped and then another table created with the same name, which of the following statements is correct? (Choose the best answer.)
   - **A.** You must rename the new table before you can flash back the dropped one.
   - **B.** You can flash back the dropped table if you specify a new name for it.

   C. You can flash back the dropped table into a different schema.
   D. You must drop the new table before flashing back the old one.

5. Under which of the following circumstances will Flashback Drop work? (Choose the best answer.)
   A. When a table has been truncated
   B. When a table has been purged
   C. When a user has been dropped
   D. When an index has been dropped
   E. None of the above

6. There are two tables in the recycle bin with the same original name. What will happen if you issue a FLASHBACK TABLE <original_name> TO BEFORE DROP command? (Choose the best answer.)
   A. The command will return an error.
   B. The oldest recycle bin table will be recovered.
   C. The newest recycle bin table will be recovered.
   D. You can't have two tables in the recycle bin with the same original name.

7. If a Flashback Table operation violates a constraint, what will happen? (Choose the best answer.)
   A. The row concerned will not be flashed back, but the rest of the operation will succeed.
   B. The flashback operation will hang until the problem is fixed.
   C. The flashback operation will be rolled back.
   D. You must disable constraints before a table flashback.

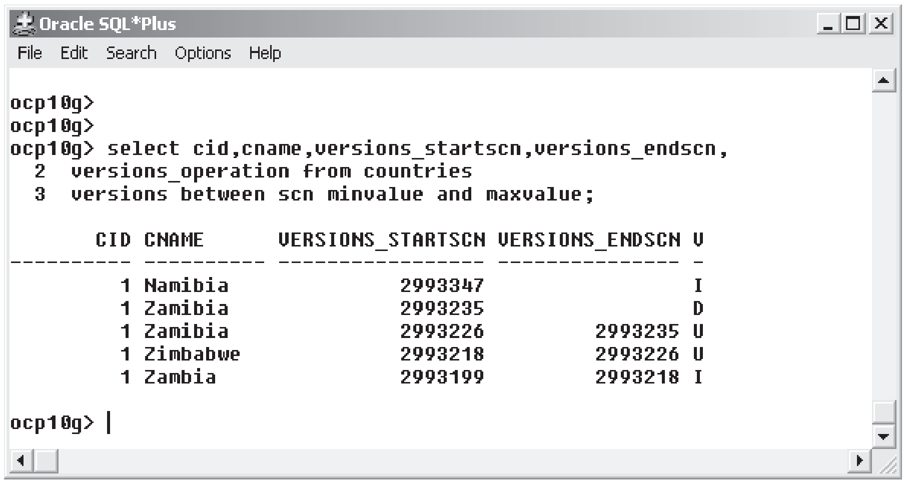
8. What is the best technique to flash back two tables in a foreign key relationship? (Choose the best answer.)
   A. Flash back the child table, and then the parent table.
   B. Flash back the parent table, and then the child table.
   C. Flash back both tables in one operation.
   D. This is not an issue; foreign key constraints are not protected by flashback.

9. Why and when must you enable row movement on a table before a flashback operation? (Choose the best answer.)
   A. Flashback Drop requires row movement, because all the rows in the table will have a different object number.
   B. Flashback Query requires row movement, because the rows will have new ROWIDs picked up from the undo segment.

C. Flashback Transaction requires row movement, because any affected rows may be moved as the transaction is reversed.

D. Flashback Table requires row movement, because any affected rows may be moved as the changes are reversed.

10. Study the following screen. Which statements are correct? (Choose two answers.)



```
ocp10g>
ocp10g>
ocp10g> select cid,cname,versions_startscn,versions_endscn,
  2  versions_operation from countries
  3  versions between scn minvalue and maxvalue;

       CID CNAME      VERSIONS_STARTSCN VERSIONS_ENDSCN V
---------- ---------- ----------------- --------------- -
         1 Namibia              2993347                 I
         1 Zamibia              2993235                 D
         1 Zamibia              2993226         2993235 U
         1 Zimbabwe             2993218         2993226 U
         1 Zambia               2993199         2993218 I

ocp10g> |
```

A. The delete of Zamibia and the insert of Namibia are uncommitted changes.

B. There cannot be a unique constraint on CID for this sequence to be possible.

C. There is currently only one row in the table.

D. If another session queried the table, it would see Zambia.

E. If another session queried the table, it would see Namibia.

11. What information can you extract from the pseudo-columns displayed by a Flashback Versions Query? (Choose all that apply.)

A. The timestamps marking the beginning and end of a version's existence

B. The SCNs marking the beginning and end of a version's existence

C. The DML operation that created the version

D. The Oracle user that executed the DML

E. The transaction that the DML was part of

F. The statement necessary to reverse the DML

G. The statement necessary to repeat the DML

12. Which of the following statements are correct about flashback, undo, and retention? (Choose two answers.)
    - **A.** Setting RETENTION GUARANTEE on a user data tablespace will ensure that a Flashback Drop will succeed, if carried out within the UNDO_RETENTION period.
    - **B.** Setting RETENTION GUARANTEE on an undo tablespace will ensure that a Flashback Query will succeed, if carried out within the UNDO_RETENTION period.
    - **C.** RETENTION GUARANTEE defaults to true for undo tablespaces, and false for user data tablespaces.
    - **D.** Flashback Database will always succeed, if the DB_FLASHBACK_RETENTION_TARGET instance parameter is less than the RETENTION_GUARANTEE period.
    - **E.** Setting the RETENTION_GUARANTEE attribute can cause DML to fail.
13. Which view will display details of all transactions, as extracted from the undo segments? (Choose the best answer.)
    - **A.** ALL_TRANSACTIONS
    - **B.** DBA_TRANSACTIONS
    - **C.** FLASHBACK_TRANSACTION_QUERY
    - **D.** UNDO_TRANSACTION_QUERY
14. Which tools can be used to flash back the drop of a table? (Choose all that apply.)
    - **A.** SQL*Plus
    - **B.** Database Control
    - **C.** RMAN
    - **D.** All of the above
15. A Flashback Table operation needs to update a row that another user is already updating. What will happen?
    - **A.** The session doing the flashback will hang until the other session commits or rolls back his change.
    - **B.** The flashback operation will not start unless it can get exclusive access to the table.
    - **C.** The flashback operation will complete, but with an error message stating that the row could not be flashed back.
    - **D.** The flashback operation will fail, with error message stating that a row or table was locked.

PART II

16. When querying DBA_OBJECTS, you see that a user owns an object whose name is BIN$Af3JifupQXuKWSpmW0gaGA==$0. What is true about this object? (Choose the best answer.)
    - **A.** It is the user's recycle bin and contains his dropped tables.
    - **B.** It is the user's recycle bin and contains his deleted rows.
    - **C.** It is a dropped table or index and will be purged automatically when its space is needed.
    - **D.** It is a dropped table or index and should be purged manually, because it is taking up space that may be needed for other objects.

Answers

1. **A, D, E, F,** and **G-b.** Of the seven technologies, five rely on undo segments and are all variations of Flashback Query.
   B-d. Flashback Database uses the flashback logs.
   C-c. Flashback Drop uses the recycle bin.
   Neither the redo log (a) nor your backups (e) have anything to do with flashback.
2. **A** and **C.** The only items that go into the recycle bin are the table and its indexes. Grants are preserved, but they remain in the data dictionary. The same is true of constraints, except for foreign key constraints, which are dropped.
3. **C.** You can query the table using its system-generated name.
4. **B.** The table must be renamed as part of the flashback to avoid a conflict.
5. **E.** The FLASHBACK DROP command can be applied only to tables, not indexes. Furthermore, the table must not have been purged. Note that you cannot flash back a TRUNCATE, because a truncation does not put anything in the recycle bin.
6. **C.** Oracle will recover the most recent table.
7. **C.** Table flashbacks are implemented as one transaction, which (like any other transaction) will be rolled back if it hits a problem. To disable constraints would be a way around the difficulty but is certainly not a requirement for flashback, and probably not a good idea.
8. **C.** The only way to guarantee success is to flash back both tables together.
9. **D.** Only Flashback Table requires row movement.
10. **C** and **E.** There is no WHERE clause in the query, so you are seeing the whole table. Versions of rows are always sorted with the newest (or current) version first, so starting from the bottom you can see that a row was inserted; updated twice; deleted; and then another inserted.

11. **A, B, C,** and **E.** The information is in the columns VERSIONS_STARTTIME, VERSIONS_ENDTIME, VERSIONS_STARTSCN, VERSIONS_ENDSCN, VERSIONS_OPERATIONS, and VERSIONS_XID. D and F can be extracted with a Flashback Transaction Query, but G cannot be done with any flashback technology.
12. **B** and **E.** Enabling the retention guarantee ensures that flashback queries will succeed, even if that means blocking DML.
13. **C.** The other views do not exist.
14. **A** and **B.** Flashback Drop can be accessed through SQL*Plus or Database Control only. RMAN can do Flashback Database, but no other flashback operations.
15. **D.** Table flashback requires row locks on every affected row, but unlike with normal DML, if the lock cannot be obtained, rather than waiting it will cancel the whole operation and roll back what work it did manage to do.
16. **C.** This is the naming convention for dropped tables and indexes, which are overwritten when the tablespace comes under space pressure.