



CHAPTER 33

Online Application Upgrades



As business needs change, the applications that support them need to change as well. Oracle provides many ways to modify existing applications; you can use features introduced with Oracle 11g to modify your applications while maintaining the high availability demanded by application users.

In this chapter you see how to minimize the impact to your applications while supporting the need for changes. A number of these approaches are described in other chapters as well.

Highly Available Databases

If an application depends on the database being available, the database architecture must support high availability throughout its design. At the database level this involves choosing from among many configuration options:

- RAC databases, where multiple instances access a single shared database, can provide transparent application failover in the event a node fails.
- Replicated databases, where separate databases serve as the “master” site for the data, provide full failover capability for both the database and the instance. Replicated databases may add complexity to your design as you decide how to resolve data-level conflicts among the separate databases.
- Standby databases, where one database is the primary database and a separate database serves as a secondary database, are available in case the primary fails.

These options all support high-availability requirements; the right choice for your environment depends on your specific application requirements. When you’re evaluating high-availability options, the evaluation criteria are usually based on disaster scenarios, the amount of lost data, and the time required to recover. However, you should also consider the more mundane and more common scenario: the need to make a change to an existing database table.

In the RAC solution, there is only one table to be changed because the separate instances share the same database. Any action that creates a DDL lock on a table affects all instances.

In a replicated environment, the objects are usually kept closely in sync; any change to one environment must be matched by the same change in the replica. Outages for DDL changes in one database are typically accompanied by outages in the replica.

In a standby solution, the standby database can be an exact physical replica or it can be a logical standby database, with different objects than the primary database. Because of this architecture, you can use a standby database to implement rolling upgrades of your application objects—first changing them in one environment and then in the second, without taking a significant outage for your application. You will need to work closely with your DBAs to configure and manage the standby environment to support this approach, but for applications that require high availability while undergoing maintenance, this architecture provides a significant benefit.

Standby databases are supported via a technology called Oracle Data Guard. Although the creation and configuration of Data Guard databases is a DBA task, developers should be aware of the architecture and features involved. For example, standby databases can be started in read-only mode to support users, then returned to standby mode. Changes to the primary database can be automatically relayed from the primary database to the standby databases with a guarantee of no data lost in the process. The standby database servers can be physically separate from the primary server.

In the following sections you see how to configure an Oracle Data Guard environment, followed by several Oracle features that minimize downtime during object changes.

Oracle Data Guard Architecture

In a Data Guard implementation, a database running in ARCHIVELOG mode is designated as the primary database for an application. One or more standby databases, accessible via Oracle Net, provide for failover capabilities. Data Guard automatically transmits redo information to the standby databases, where it is applied. As a result, the standby databases are transactionally consistent. Depending on how you configure the redo application process, the standby databases may be in sync with the primary database or may lag behind it.

The redo log data is transferred to the standby databases automatically as defined via your initialization parameter settings. On the standby server, the redo log data is received and applied.

Physical vs. Logical Standby Databases

Two types of standby databases are supported: physical standbys and logical standbys. A *physical standby* database has the same structures as the primary database. A *logical standby* database may have different internal structures (such as additional indexes used for reporting). You synchronize a logical standby database with the primary by transforming the redo data into SQL statements that are executed against the standby database.

Physical and logical standby databases serve different purposes. A physical standby database is a block-for-block copy of the primary database, so it can be used for database backups in place of the primary database. During disaster recovery, the physical standby looks exactly like the primary database it replaces.

A logical standby database, because it supports additional database structures, can more easily be used to support specific reporting requirements that would otherwise burden the primary database. Additionally, rolling upgrades of primary and standby databases can be performed with minimal downtime when logical standbys are used. The type of standby to use depends on your needs; many environments start out using physical standby databases for disaster recovery and then add in other logical standby databases to support specific reporting and business requirements.



NOTE

The operating system and platform architecture on the primary and standby locations must be identical. The directory structures and operating system patch levels may differ, but you should minimize the differences to simplify administration and failover processes. If the standby is located on the same server as the primary, you must use a different directory structure for the two databases, and they cannot share an archive log directory.

Data Protection Modes

When you configure the primary and standby databases, you will need to determine the level of data loss that is acceptable to the business. In the primary database, you will define the archive log destination areas, at least one of which will refer to the remote site used by the standby database. The ASYNC, SYNC, ARCH, LGWR, NOAFFIRM, and AFFIRM attributes of the LOG_ARCHIVE_DEST_ *n* parameter setting for the standby database will direct Oracle Data Guard to select among several modes of operation:

- In maximum protection (or “no data loss”) mode, at least one standby location must be written to before a transaction commits in the primary database. The primary database shuts down if the standby database’s log location is unavailable.
- In maximum availability mode, at least one standby location must be written to before a transaction commits in the primary database. If the standby location is not available, the

538 Part IV: PL/SQL

primary database does not shut down. When the fault is corrected, the redo generated since the fault is transported and applied to the standby databases.

- In maximum performance mode (the default), transactions can commit before their redo information is sent to the standby locations. Commits in the primary database occur as soon as writes to the local online redo logs complete.

Once you have decided the type of standby and the data protection mode for your configuration, you can create your standby database.

Creating the Standby Database Configuration

You can use SQL*Plus, Oracle Enterprise Manager (OEM), or Data Guard–specific tools to configure and administer Data Guard configurations. The parameters you set will depend on the configuration you choose.

If the primary and standby databases are on the same server, you will need to set a value for the `DB_UNIQUE_NAME` initialization parameter. Because the directory structures for the two databases will be different, you must either manually rename files or define values for the `DB_FILE_NAME_CONVERT` and `LOG_FILE_NAME_CONVERT` parameters in the standby database. You must set up unique service names for the primary and standby databases via the `SERVICE_NAMES` initialization parameter.

If the primary and standby databases are on separate servers, you can use the same directory structures for each, avoiding the need for the filename conversion parameters. If you use a different directory structure for the database files, you will need to define the values for the `DB_FILE_NAME_CONVERT` and `LOG_FILE_NAME_CONVERT` parameters in the standby database.

In physical standby databases, all the redo comes from the primary database. When physical standby databases are opened in read-only mode, no redo is generated. Oracle Data Guard does, however, use archived redo log files to support the replication of the data and SQL commands used to update the standby databases.

NOTE

For each standby database, you should create a standby redo log file to store redo data received from the primary database.

Creating Logical Standby Databases

Logical standby databases follow many of the same steps used to create physical standby databases. Because they rely on the reexecution of SQL commands, logical standby databases have greater restrictions on their use. If any of your tables in the primary database use the following datatypes, they will be skipped during the redo application process:

- BFILE
- ROWID
- UROWID
- User-defined datatypes
- Object types
- REFS
- Varying arrays

- Nested tables
- XMLtype

**NOTE**

Encrypted columns are not supported in a logical standby database.

Additionally, tables that use table compression and the schemas that are installed with the Oracle software are skipped during redo application. The `DBA_LOGSTDBY_UNSUPPORTED` view lists the objects that are not supported for logical standby databases. The `DBA_LOGSTDBY_SKIP` view lists the schemas that will be skipped.

A logical standby database is not identical to the primary database. Each transaction that is executed in the logical standby database must be the logical equivalent of the transaction that was executed in the primary database. Therefore, you should make sure your tables have the proper constraints on them—primary keys, unique constraints, check constraints, and foreign keys—so the proper rows can be targeted for update in the logical standby database. You can query `DBA_LOGSTDBY_NOT_UNIQUE` to list tables that lack primary keys or unique constraints in the primary database.

Using real-time apply

By default, redo data is not applied to a standby database until the standby redo log file is archived. When you use the real-time apply feature, redo data is applied to the standby database as it is received, thus reducing the time lag between the databases and potentially shortening the time required to fail over to the standby database.

To enable real-time apply in a physical standby database, execute the following command in the standby database:

```
alter database recover managed standby database
using current logfile;
```

For a logical standby database, the command to use is

```
alter database start logical standby apply immediate;
```

The `Recovery_Mode` column of the `V$ARCHIVE_DEST_STATUS` view will have a value of `MANAGED REAL-TIME APPLY` if real-time apply has been enabled.

As shown earlier in this chapter, you can enable the redo application on a physical standby database via the command

```
alter database recover managed standby database
disconnect from session;
```

The **disconnect from session** clause allows the command to run in the background after you disconnect from your Oracle session. When you start a foreground session and issue the same command without the **disconnect from session** clause, control is not returned to the command prompt until the recovery is cancelled by another session. To stop the redo application in a physical standby database—whether in a background session or a foreground session—use the following command:

```
alter database recover managed standby database cancel;
```

540 Part IV: PL/SQL

For a logical standby database, the command to stop the log apply services is

```
alter database stop logical standby apply;
```

Managing Roles—Switchovers and Failovers

Each database that participates in a Data Guard configuration has a role—it is either a primary database or a standby database. At some point, those roles may need to change. For example, if there is a hardware failure on the primary database's server, the primary database may fail over to the standby database. Depending on your configuration choices, there may be some loss of data during a failover.

A second type of role change is called a *switchover*. This occurs when the primary database switches roles with a standby database, and the standby becomes the new primary database. During a switchover, no data should be lost. Switchovers and failovers require manual intervention by a database administrator.

Switchovers

Switchovers are planned role changes, usually to allow for maintenance activities to be performed on the primary database server. A standby database is chosen to act as the new primary database, the switchover occurs, and applications now write their data to the new primary database. At some later point in time you can switch the databases back to their original roles.

NOTE

You can perform switchovers with either a logical standby database or a physical standby database; the physical standby database is the preferred option.

What if you have defined multiple standby databases? When one of the physical standby databases becomes the new primary database, the other standby databases must be able to receive their redo log data from the new primary database. In that configuration, you must define the LOG_ARCHIVE_DEST_n parameters to allow those standby sites to receive data from the new primary database location.

NOTE

Verify that the database that will become the new primary database is running in ARCHIVELOG mode.

The following sections detail the steps required to perform a switchover to a standby database. The standby database should be actively applying redo log data prior to the switchover, because this will minimize the time required to complete the switchover.

Switchovers to Physical Standby Databases

Switchovers are initiated on the primary database and completed on the standby database. In this section, you will see the steps for performing a switchover to a physical standby database.

Begin by verifying that the primary database is capable of performing a switchover. Query V\$DATABASE for the value of the Switchover_Status column:

```
select Switchover_Status from V$DATABASE;
```

Chapter 33: Online Application Upgrades **541**

If the `Switchover_Status` column's value is anything other than `TO STANDBY`, it is not possible to perform the switchover (usually due to a configuration or hardware issue). If the column's value is `SESSIONS ACTIVE`, you should terminate active user sessions. Valid values for the `Switchover_Status` column are shown in Table 33-1.

From within the primary database, you can initiate its transition to the physical standby database role with the following command:

```
alter database commit to switchover to physical standby;
```

As part of executing this command, Oracle will back up the current primary database's controlfile to a trace file. At this point, you should shut down the primary database and mount it:

```
shutdown immediate;
startup mount;
```

The primary database is prepared for the switchover; you should now go to the physical standby database that will serve as the new primary database.

In the physical standby database, check the switchover status in the `V$DATABASE` view; its status should be `TO PRIMARY` (see Table 33-1). You can now switch the physical standby database to the primary via the following command:

```
alter database commit to switchover to primary;
```

Switchover_Status Value	Description
NOT ALLOWED	The current database is not a primary database with standby databases.
PREPARING DICTIONARY	This logical standby database is sending its redo data to a primary database and other standby databases to prepare for the switchover.
PREPARING SWITCHOVER	Used by logical standby configurations while redo data is being accepted for the switchover.
RECOVERY NEEDED	This standby database has not received the switchover request.
SESSIONS ACTIVE	There are active SQL sessions in the primary database; they must be disconnected before continuing.
SWITCHOVER PENDING	Valid for standby databases in which the primary database switchover request has been received but not processed.
SWITCHOVER LATENT	The switchover did not complete and went back to the primary database.
TO LOGICAL STANDBY	This primary database has received a complete dictionary from a logical standby database.
TO PRIMARY	This standby database can switch over to a primary database.
TO STANDBY	This primary database can switch over to a standby database.

TABLE 33-1 *Switchover_Status Values*

542 Part IV: PL/SQL

Shut down and start up the database, and it will complete its transition to the primary database role. After the database starts, force a log switch via the **alter system switch logfile** command and then start the redo apply services on the standby databases if they were not already running in the background.

Switchovers to Logical Standby Databases

Switchovers are initiated on the primary database and completed on the standby database. In this section, you will see the steps for performing a switchover to a logical standby database.

Begin by verifying that the primary database is capable of performing a switchover. Query V\$DATABASE for the value of the Switchover_Status column:

```
select Switchover_Status from V$DATABASE;
```

For the switchover to complete, the status must be either TO STANDBY, TO LOGICAL STANDBY, or SESSIONS ACTIVE.

In the primary database, issue the following command to prepare the primary database for the switchover:

```
alter database prepare to switchover to logical standby;
```

In the logical standby database, issue the following command:

```
alter database prepare to switchover to primary;
```

At this point, the logical standby database will begin transmitting its redo data to the current primary database and to the other standby databases in the configuration. The redo data from the logical standby database is sent but is not applied at this point.

In the primary database, you must now verify that the dictionary data was received from the logical standby database. The Switchover_Status column value in V\$DATABASE must read TO LOGICAL STANDBY in the primary database before you can continue to the next step. When that status value is shown in the primary database, switch the primary database to the logical standby role:

```
alter database commit to switchover to logical standby;
```

You do not need to shut down and restart the old primary database. You should now go back to the original logical standby database and verify its Switchover_Status value in V\$DATABASE (it should be TO PRIMARY). You can then complete the switchover; in the original logical standby database, issue the following command:

```
alter database commit to switchover to primary;
```

The original logical standby database is now the primary database. In the new primary database, perform a log switch:

```
alter system archive log current;
```

In the new logical standby database (the old primary database), start the redo apply process:

```
alter database start logical standby apply;
```

The switchover is now complete.

Making Low-Impact DDL Changes

If you only have one database available, you need to be careful when making changes to your tables and indexes. When you create or modify an object, you must first acquire a DDL lock on the object to prevent other types of access against the object. Because of this locking issue, you will need to avoid changing objects while users are accessing them. Furthermore, changing an object will invalidate the stored procedures that reference the object.

Although you should ideally make changes to objects at times when users are not accessing them, your maintenance time windows may not be sufficiently large enough to support that requirement. You should always plan for enough time to make the change, test the change, and recompile any objects that the change impacts. In the following sections you will see methods for making changes in ways that minimize the size of the maintenance window needed.

Creating Virtual Columns

As of Oracle 11g, you can create virtual columns in tables rather than storing derived data. You may have a virtual column that is based on other values in the same row (adding two columns together, for instance). In Oracle 11g, you can specify the function as part of the table definition, which allows you to index the virtual column and partition the table by it.

From a change perspective, the benefit to this feature is that you can avoid creating physical columns in the database. Rather than trying to add a column of derived data to a very large table, you can create a virtual column—and the data will be available to the users even though you have not executed a very time-consuming DDL command.

Consider the AUTHOR table:

```
create table AUTHOR
(AuthorName VARCHAR2(50) primary key,
Comments VARCHAR2(100));
```

When values are inserted into AUTHOR, they are inserted in uppercase:

```
insert into AUTHOR values
('DIETRICH BONHOEFFER', 'GERMAN THEOLOGIAN, KILLED IN A WAR CAMP');

insert into AUTHOR values
('ROBERT BRETALL', 'KIERKEGAARD ANTHOLOGIST');
```

You can create an additional mixed-case column using the **generated always** clause to tell Oracle to create the virtual column:

```
create table AUTHOR2
(AuthorName VARCHAR2(50) primary key,
Comments VARCHAR2(100),
MixedName VARCHAR2(50)
generated always as
(initcap(AuthorName) ) virtual
);
```

The code used to create the virtual column can be complex, including **case** statements and other functions. Note that you do not have to create triggers or other mechanisms to populate the column; Oracle maintains it as a virtual value for each row in the table. The value is not physically stored with the table; it is generated when needed.

544 Part IV: PL/SQL

**NOTE**

Attempts to provide a value for the virtual column during inserts will result in an error.

You can create an index on the virtual column as if it were a standard column in the table. Additionally, you can partition the table based on this column (as described later in this chapter).

Altering Actively Used Tables

When you issue the **alter table** command, Oracle attempts to acquire a DDL lock on the table. If anyone else is accessing the table at that time, your command will fail—you need to have exclusive access to the table while you are changing its structure. You may need to try repeatedly to execute your command in order to acquire the lock you need.

As of Oracle 11g, you can use the DDL lock timeout options to work around this problem. You can execute the **alter session** command to set a value for the **ddl_lock_timeout** parameter, specifying the number of seconds during which Oracle should continually retry your command. The retry attempts will continue until the command is successful or the timeout limit is reached, whichever comes first.

To try your command for 60 seconds, issue the following command:

```
alter session set ddl_lock_timeout=60;
```

DBAs can enable this at the database level via the **alter system** command, as shown here:

```
alter system set ddl_lock_timeout=60;
```

Adding NOT NULL Columns

In Oracle 11g, you can add NOT NULL columns to tables that already contain rows. To do so, you must specify a default value for the new column. For very large tables, this can present a problem because millions of rows may need to be updated to contain the new column value. Oracle 11g automatically handles this in a way that avoids significant impacts to the users.

Let's consider again an attempt to add just the Condition column to the TROUBLE table:

```
alter table TROUBLE add (Condition VARCHAR2(9) NOT NULL);
```

If TROUBLE already has rows, the Condition column can't be added because the existing rows would have NULL values for it. The solution is to add a DEFAULT constraint for the Condition column:

```
alter table TROUBLE add (Condition VARCHAR2(9)
    default 'SUNNY' NOT NULL);
```

Oracle will now create the new column, but it will *not* update the existing rows. If a user selects a row that has a null value in the Condition column, the default value will be returned. Oracle's approach solves two problems: It lets you add a NOT NULL column, and it avoids having to update every existing row. In large tables, that can represent a significant savings in terms of the space required to support the transaction size.

Online Object Reorganizations

Several online options are available when moving an object online. For indexes, you can use the **alter index rebuild online** clause to modify an index while its base table is being accessed. You can use the **alter index rebuild online** command to move the partitions of an existing index.

Chapter 33: Online Application Upgrades **545**

For index-organized tables, you can use the **move online** option of the **alter table** command.

For partitioned tables, you can move partitions via the **partition** clauses of the **alter table** command.

You can perform online reorganizations of most types of tables via the DBMS_REDEFINITION package. The DBMS_REDEFINITION approach creates a replica of your table and creates triggers and logs to keep the replica in sync with the original table (as if it were a materialized view). The movement of the table data can take place any time you want; you can execute manual refreshes of the data in the replica. When you are ready to apply the change to your system, the FINISH_REDEF_TABLE completes the redefinition process and the replica takes the place of the original table. Note that the original table will still exist and will still have all its original data until you manually drop it.

The redefinition process has five steps:

1. Verify the table can be rebuilt online.
2. Create an interim table and its indexes, grants, constraints, and triggers.
3. Start the redefinition.
4. Optionally sync the source/destination tables.
5. Abort or finish the redefinition.

The following steps show the redefinition of the PRACTICE.EMP table from the USERS to the USERS_DEST tablespace. In the process, we'll also partition the EMP table on the fly.

Verify the Table Can Be Rebuilt Online

To verify the table can be rebuilt online, execute the CAN_REDEF_TABLE procedure, providing the schema owner and table name as input:

```
execute DBMS_REDEFINITION.CAN_REDEF_TABLE('SCOTT','EMP');
```

If an error stack is reported, the first error listed will tell you what the problem is; later errors concerning the DBMS_REDEFINITION package can be ignored.

Create the Interim Table

What do you want EMP to look like in the new tablespace? Create an interim table that will be used during the redefinition process. To simplify matters, keep the column names and order the same as the source EMP table.

```
create table EMP_DEST
(EMPNO    NUMBER(4) PRIMARY KEY,
 ENAME    VARCHAR2(10),
 JOB      VARCHAR2(9),
 MGR      NUMBER(4),
 HIREDATE DATE,
 SAL      NUMBER(7,2),
 COMM     NUMBER(7,2),
 DEPTNO   NUMBER(2))
partition by range (DeptNo)
(partition PART1
 values less than ('30'),
 partition PART2
 values less than (MAXVALUE))
tablespace USERS_DEST;
```

546 Part IV: PL/SQL

In this example, the EMP_DEST table will serve as the interim table during the redefinition. Once the process is complete, EMP_DEST will replace the old EMP table—and the only indexes, constraints, triggers, and grants available for EMP then will be those you create on EMP_DEST now. You can optionally create the indexes, triggers, grants, and constraints after the redefinition process completes, but creating them now avoids the need for DDL locks on active tables later.

You can execute the COPY_TABLE_DEPENDENTS procedure to create on the target (EMP_DEST) all the indexes, triggers, constraints, privileges, and statistics that exist on the source. The format for COPY_TABLE_DEPENDENTS is shown in the following listing:

```
DBMS_REDEFINITION.COPY_TABLE_DEPENDENTS (
  uname           IN  VARCHAR2,
  orig_table      IN  VARCHAR2,
  int_table       IN  VARCHAR2,
  copy_indexes    IN  PLS_INTEGER := 1,
  copy_triggers   IN  BOOLEAN     := TRUE,
  copy_constraints IN  BOOLEAN     := TRUE,
  copy_privileges IN  BOOLEAN     := TRUE,
  ignore_errors   IN  BOOLEAN     := FALSE,
  num_errors      OUT PLS_INTEGER,
  copy_statistics IN  BOOLEAN     := FALSE,
  copy_mvlog      IN  BOOLEAN     := FALSE);
```

Start the Redefinition

The START_REDEF_TABLE procedure begins the redefinition. The EMP_DEST table will now be populated with the committed data in the EMP table, so the time and undo requirements for this step depend on the size of the EMP table. Pass the schema owner, old table name, and the interim table name as parameters:

```
execute DBMS_REDEFINITION.START_REDEF_TABLE -
  ('SCOTT', 'EMP', 'EMP_DEST');
```

If EMP_DEST had a different column list than EMP, that column list would be passed as the fourth parameter to the START_REDEF_TABLE procedure. Once this procedure has completed, you can query EMP_DEST to verify its data contents.

Abort the Process (Optional)

If you need to abort the redefinition process at this point, use the ABORT_REDEF_TABLE procedure, with the schema owner, source table, and interim table as input parameters:

```
execute DBMS_REDEFINITION.ABORT_REDEF_TABLE -
  ('SCOTT', 'EMP', 'EMP_DEST');
```

After aborting the redefinition, you should consider truncating the interim (EMP_DEST) table as well.

Sync the Tables (Optional)

During the conclusion of the redefinition process, Oracle will sync the data between the source and interim tables. To shorten the time required by that part of the process, you can sync the tables prior to the final step. This optional step allows you to instantiate the latest production data in the interim table, minimizing later impact on your online users.

To sync the source and interim tables, use the SYNC_INTERIM_TABLE procedure, with the schema owner, source table, and interim table as input parameters:

```
execute DBMS_REDEFINITION.SYNC_INTERIM_TABLE -
('SCOTT', 'EMP', 'EMP_DEST');
```

Finish the Redefinition Process

To complete the redefinition process, execute the FINISH_REDEF_TABLE procedure, as shown in the following examples. The input parameters are the owner, source table, and interim table names.

```
execute DBMS_REDEFINITION.FINISH_REDEF_TABLE -
('SCOTT', 'EMP', 'EMP_DEST');
```

Verify the Redefinition

Because the EMP table should now be partitioned and moved to USERS_DEST, you can verify its redefinition by querying DBA_TAB_PARTITIONS:

```
select Table_Name, Tablespace_Name, High_Value
       from DBA_TAB_PARTITIONS
       where Owner = 'SCOTT';
```

TABLE_NAME	TABLESPACE_NAME	HIGH_VALUE
EMP	USERS_DEST	MAXVALUE
EMP	USERS_DEST	'30'

As shown in the preceding listing, the EMP table partitions both reside in the USERS_DEST tablespace. At this point, you should verify that the foreign keys on EMP are enabled, that all necessary grants are in place, that the indexes are all in place, and that all triggers are enabled.

Dropping a Column

You can use the table reorganization options to drop columns (because the source and target can have different column definitions). As an alternative, you can mark columns to have an “unused” state during regular usage, then drop them when a longer maintenance window is available.

For example, you can mark the Wind column as unused:

```
alter table TROUBLE set unused column Wind;
```

Marking a column as “unused” does not release the space previously used by the column until you drop the unused columns:

```
alter table TROUBLE drop unused columns;
```

You can query USER_UNUSED_COL_TABS, ALL_UNUSED_COL_TABS, and DBA_UNUSED_COL_TABS to see all tables with columns marked as unused.

NOTE

Once you have marked a column as “unused,” you cannot access that column.

548 Part IV: PL/SQL

You can drop multiple columns in a single command, as shown in the following listing:

```
alter table TROUBLE drop (Condition, Wind);
```

**NOTE**

*When dropping multiple columns, you should not use the **column** keyword of the **alter table** command; it causes a syntax error. The multiple column names must be enclosed in parentheses, as shown in the preceding listing.*

If the dropped columns are part of primary keys or unique constraints, you will need to also use the **cascade constraints** clause as part of your **alter table** command. If you drop a column that belongs to a primary key, Oracle will drop both the column and the primary key index.