PREV
13 Manipulating Large Data Sets

NEXT
A About the Download

14

Controlling User Access

CERTIFICATION OBJECTIVES

This chapter explores the subject of user access and the privileges associated with performing actions in the database. Every action performed by any user account requires a corresponding privilege or set of privileges to perform that action. There are two categories of privileges. *System* privileges are required to perform a task in the database; *object* privileges are required to use those system privileges on any given database object in particular. Privileges may be granted to a user account or to another database object called a *role*. A role, in turn, can be granted to a user account, which effectively grants the set of privileges collected within the role. Once granted, privileges and roles can later be revoked. Together, privileges and roles are the mechanism for managing and controlling access to the database by user accounts. This chapter looks at how to create and manage privileges and roles.

A word of warning about the sample code contained in this chapter: some of it has the ability to change your database permanently with results that may be undesirable. Some of our code samples will look at SQL code that uses the SYSTEM user account, an important account that should be controlled by experienced database administrators in any production database. You should always check with your database administrator (DBA) before trying any code samples from any book, but this chapter in particular includes code that you should not execute in a professional installation without first checking with your DBA.

CERTIFICATION OBJECTIVE 14.01

DIFFERENTIATE SYSTEM PRIVILEGES FROM OBJECT PRIVILEGES

Throughout this book, we've looked at how a user account can use SQL statements to create and use a variety of database objects. However, before any user account can execute a SQL statement, it must be granted the privilege to execute that SQL statement. Furthermore, once a database object has been created, any user account that will use the database object must first be granted privileges to do so.

There are three general categories of privileges, as described in Table 14-1.

TABLE 14-1 Types of Privileges

Type of Privilege	Description
System privilege	The ability to perform a particular task in the database
Object privilege	The ability to perform a particular task on a particular database object
Role	A collection of one or more system privileges and/or object privileges and/or other roles

We'll review each of the items listed in Table 14-1 in this chapter.

System Privileges

privilege CREATE SESSION. To create a table, a user account must be granted the system privilege CREATE TABLE.

There are more than 100 different system privileges that can be granted to a user account. Table 14-2 lists some of the system privileges that are required to perform the tasks we've discussed in this book.

TABLE 14-2 Some System Privileges

System Privilege	Description
CREATE SESSION	Connect to the database.
CREATE TABLE	Create a table in your user account. Includes the ability to use ALTER and DROP TABLE. Also includes the ability to use CREATE, ALTER, and DROP INDEX on objects.
CREATE VIEW	Create a view in your user account. Includes ALTER and DROP.
CREATE SEQUENCE	Create a sequence in your user account. Includes ALTER and DROP.
CREATE SYNONYM	Create a synonym in your user account. Includes ALTER and DROP. Does not include PUBLIC synonyms (see CREATE PUBLIC SYNONYM).
CREATE ROLE	Create a role. Includes ALTER and DROP.
CREATE PUBLIC SYNONYM	Create a synonym in the PUBLIC account. Does not include DROP, which is separate.
DROP PUBLIC SYNONYM	Drop a synonym from the PUBLIC account.
CREATE ANY TABLE	Create a table within any user account.
ALTER ANY TABLE	Alter a table within any user account.
DELETE ANY TABLE	Delete from any table within any user account.
DROP ANY TABLE	Drop or truncate any table within any user account.
INSERT ANY TABLE	Insert into any table within any user account.
SELECT ANY TABLE	Select from any table within any user account.
UPDATE ANY TABLE	Update any table within any user account.
CREATE ANY VIEW	Create a view in any user account.
DROP ANY VIEW	Drop a view from any user account.
CREATE ANY INDEX	Create an index in any user account.
ALTER ANY INDEX	Alter an index in any user account.
DROP ANY INDEX	Drop an index from any user account.
CREATE ANY SEQUENCE	Create a sequence in any user account.
ALTER ANY SEQUENCE	Alter a sequence in any user account.
DROP ANY SEQUENCE	Drop a sequence from any user account.
SELECT ANY SEQUENCE	Select from a sequence in any user account.
CREATE ANY SYNONYM	Create a synonym in any user account.
DROP ANY SYNONYM	Drop a synonym from any user account.
CREATE ANY DIRECTORY	Create a directory in any user account.
DROP ANY DIRECTORY	Drop a directory from any user account.
ALTER ANY ROLE	Alter a role in the database.
DROP ANY ROLE	Drop any role in the database.
GRANT ANY ROLE	Grant any role in the database.
FLASHBACK ANY TABLE	Perform flashback operations on any table in the database.
CREATE USER	Create a user account.
ALTER USER	Alter a user account.
DROP USER	Drop a user account.
GRANT ANY PRIVILEGE	Grant any system privilege to any user account in the database.
GRANT ANY OBJECT PRIVILEGE	Grant, to any other user account in the database, any object privilege that the object's owner is also able to grant.

System privileges differ from object privileges in that system privileges are what a user account must have to create database objects, among other things. Then, once a database object has been created, object privileges on that database object can be granted to other users.

For example, the right to execute the SQL statement CREATE TABLE and create a new database table is a system privilege. But the ability to change rows of data in, for example, a table called BENEFITS owned by a user account named EUNICE is an object privilege. In other words, an object privilege is the right to do something to a particular object.

As an analogy, consider the concept of a driver's license. A driver's license is sort of like a system privilege; it's the right to drive a car in a general sense. Once you have a driver's license, if you get a car, you can drive it. But you don't have the right to drive anyone's car in particular unless the owner specifically authorizes you to do so.

The right to drive someone else's car is like an object privilege. You need both of these privileges in order to drive a car and to be in full compliance with the law. The same is true in the database: you need system privileges to perform particular tasks, and you need object privileges to perform those tasks on an object in particular.

Let's look at some of the syntax for granting privileges. Note that for some of the upcoming examples, we'll use the SQL*Plus tool and some SQL*Plus commands. These SQL*Plus commands do not require the semicolon termination character that is required in SQL statements.

We'll use the SQL*Plus command CONNECT to log in to another user account. You can also use the SQL*Plus command SHOW USER to confirm which account is currently active in the session. SQL*Plus commands are helpful to use in your SQL sessions. But they are not on the exam.

Prerequisites

Before we get started with GRANT and REVOKE statements, let's review some supporting statements that aren't specifically included in the exam objectives but are useful for demonstrating system privileges, object privileges, roles, and their capabilities.

CREATE, ALTER, and DROP USER

Let's look at how to create a user account. Any SQL user with the CREATE USER system privilege may execute the CREATE USER statement, whose syntax looks like this:

```
CREATE USER username IDENTIFIED BY password;
```

In this statement, *username* is a name you specify according to the rules for naming database objects. The *password* follows the same rules. (Note that passwords are case sensitive by default starting with Oracle 11*g*.)

For example, this statement will create a user name JOAN with the password DEMERY:

```
CREATE USER JOAN IDENTIFIED BY DEMERY;
```

You can use the ALTER USER statement to change the password, like this:

Finally, you can remove a user from the database using the DROP USER statement, like this:

```
DROP USER username;
```

If a user account owns any database objects, the preceding statement won't work, and you'll need to use this:

```
DROP USER username CASCADE;
```

The CASCADE option directs SQL to drop the user account and all of the objects it owns.

Once a user object has been created, it can be granted privileges, as you'll see in an upcoming section.

CONNECT

The CONNECT statement is not a SQL statement but a SQL*Plus enhancement you can use within the Oracle SQL*Plus tool. Once you've started SQL*Plus, you can use CONNECT to log in or switch login sessions from one user account to another. If, for example, you are using the SQL*Plus tool and have logged in to the EFCODD account and created the user account JOAN, you can log in to the JOAN account directly from EFCODD with this statement:

```
CONNECT JOAN/HAWAII
```

This assumes the user account JOAN is still using the password HAWAII. It also assumes that JOAN has been granted the minimum system privileges to log in, such as CREATE SESSION.

Again, a semicolon termination character is not required in SQL*Plus statements. It is accepted but not required. The semicolon termination character is required in SQL statements but is optional in SQL*Plus statements.

Tablespaces

In the course of setting up a new user account, the topic of tablespaces must be addressed. However, the topic of tablespaces goes beyond our scope and is not included in the exam objectives, so we'll show the simple way to address the tablespace requirement, as follows:

```
GRANT UNLIMITED TABLESPACE TO username;
```

This would probably not be something that your typical production DBA would do. Tablespaces are controlled by database administrators. A typical DBA generally creates uniquely named tablespaces and carefully allocates space quotas to them. We, however, aren't concerned with any of that for this book or for the exam. So for us, the preceding statement is what we'll include. If you want to learn more, we encourage you to check out any of the outstanding books from Oracle Press on the topic of database administration. In the meantime, if you're working on your own test system on your own personal machine, this particular statement that grants UNLIMITED TABLESPACE is more than adequate for our purposes going forward. If you're using these at work, check with your DBA before trying any of the code samples in this chapter.

GRANT and REVOKE

Now let's get down to business. System privileges are granted with the GRANT statement. Here's an example of a SQL session that logs in to the Oracle SYSTEM account, creates a new user, and grants the new user account some initial system privileges using three GRANT statements (line numbers added):

```
01   CONNECT SYSTEM/MANAGER
02   CREATE USER HAROLD IDENTIFIED BY LLOYD;
03   GRANT CREATE SESSION TO HAROLD;
04   GRANT UNLIMITED TABLESPACE TO HAROLD;
05   GRANT CREATE TABLE TO HAROLD;
```

In these statements, here is what we are doing:

Line 1 We establish a user session with the user account SYSTEM, with a password of MANAGER. The SYSTEM account is installed with every Oracle database, and the DBA installing Oracle assigns the password. (Warning: Do not try this on a production system. No self-respecting production system should have a SYSTEM account password set to a value of MANAGER anyway, but the point is that if you have installed your own version of the Oracle database on your own local machine and it is not used for production work, then you can try this, but if you're trying things out within a system at your workplace or somewhere comparable, then be sure to check with your database administrator before trying this.)

Line 2 We create a new user account called HAROLD, with the password LLOYD.

Line 3 We use the SQL statement GRANT to give the CREATE SESSION privilege to user HAROLD. This is a minimum requirement for us to be able to log in to the database with the HAROLD user account; without this GRANT statement, we couldn't successfully log in with the user account HAROLD.

Line 4 This is one way to ensure that HAROLD can create objects. See our earlier discussion about tablespaces in the previous section.

Line 5 Using GRANT, we give the system privilege CREATE TABLE to user account HAROLD.

See Figure 14-1 for the results of these statements in the SQL*Plus

```
SQL> CONNECT SYSTEM/MANAGER
Connected.
SQL> CREATE USER HAROLD IDENTIFIED BY LLOYD;

User created.

SQL> GRANT CREATE SESSION TO HAROLD;

Grant succeeded.

SQL> GRANT UNLIMITED TABLESPACE TO HAROLD;

Grant succeeded.

SQL> GRANT CREATE TABLE TO HAROLD;

Grant succeeded.

SQL>
```

Now let's log in to HAROLD and try out what we've done. For that, we'll try the following SQL statements:

```
CONNECT HAROLD/LLOYD
CREATE TABLE CLOCKTOWER (CLOCK_ID NUMBER(11));
CREATE SEQUENCE SEQ_CLOCK_ID;
```

See Figure 14-2 for the results. Note that we aren't able to create the sequence because we haven't been granted sufficient privileges to do so. For that, we'll need to log back in to the SYSTEM account and grant the system privilege CREATE SEQUENCE to HAROLD. Once that has been accomplished, we can log back in to HAROLD and create the sequence (see Figure 14-3).

FIGURE 14-2 SQL*Plus session: testing system privileges

```
SQL> CONNECT HAROLD/LLOYD
Connected.
SQL> CREATE TABLE CLOCKTOWER (CLOCK_ID NUMBER(11));

Table created.

SQL> CREATE SEQUENCE SEQ_CLOCK_ID;
CREATE SEQUENCE SEQ_CLOCK_ID
*
ERROR at line 1:
ORA-01031: insufficient privileges

SQL>
```

FIGURE 14-3 SQL*Plus session: creating the sequence

```
SQL> CONNECT SYSTEM/MANAGER
Connected.
SQL> GRANT CREATE SEQUENCE TO HAROLD;

Grant succeeded.

SQL> CONNECT HAROLD/LLOYD
Connected.
SQL> CREATE SEQUENCE SEQ_CLOCK_ID;

Sequence created.

SQL>
```

In these examples, we have been logged in to the SYSTEM account to grant these privileges, but any qualified DBA account will do and is preferable in any serious installation with multiple Oracle users. In such a situation, the less time a developer or DBA spends in the SYSTEM account—or the other restricted default DBA accounts in the Oracle database such as SYS—the less likely a mistake will accidentally cause some serious damage to the database.

The basic syntax for the GRANT statement is simple.

```
GRANT privilege TO user options;
```

Here, *privilege* is one of the several dozens of system privileges that are already defined in the database (see the *Oracle Database SQL Language Reference Manual* for a complete list). Multiple privileges can be granted at once by separating each additional privilege with a comma, as in GRANT *privilege, privilege*. (We'll discuss *option* in an upcoming section.)

The basic syntax for REVOKE is comparable.

```
REVOKE privilege FROM user;
```

Note that you grant TO and you revoke FROM.

Once a system privilege is revoked from a user, the effect is immediate. However, any actions taken prior to the revocation stand. In other words, if a user account has been granted the system privilege CREATE TABLE and then creates some tables but then has the CREATE TABLE system privilege revoked, the created tables already in existence remain in place. They do not disappear. But the owning user may not create additional tables while the CREATE TABLE system privilege is revoked.

We've looked at a few system privileges, and we've said that they are somewhat like a driver's license. Now let's extend the analogy a little bit: imagine what would happen if you could get a universal driver's license that carried with it the ability to drive anyone's car legally without the car's owner express permission. Such a concept exists within the Oracle database, and it's embodied in the keyword ANY. Let's look at that next.

ANY

Some system privileges include the keyword ANY in the title. For example, there is a system privilege CREATE ANY TABLE, which is the ability to create a table in any user account anywhere in the database. Let's look at a sample session that involves this privilege.

```
CONNECT SYSTEM/MANAGER

CREATE USER LAUREL IDENTIFIED BY POKE;
GRANT CREATE SESSION TO LAUREL;
GRANT UNLIMITED TABLESPACE TO LAUREL;
GRANT CREATE TABLE TO LAUREL;

CREATE USER HARDY  IDENTIFIED BY CLOBBER;
GRANT CREATE SESSION TO HARDY;
GRANT UNLIMITED TABLESPACE TO HARDY;
GRANT CREATE ANY TABLE TO HARDY;

CONNECT LAUREL / POKE
CREATE TABLE MOVIES (MOVIE_ID NUMBER(7));

CONNECT HARDY / CLOBBER
CREATE TABLE LAUREL.TVSHOWS (TVSHOW_ID NUMBER(7));
```

The result of the preceding SQL statements is that two user accounts will be created. Also, two tables will be created, with one table called MOVIES and another table called TVSHOWS. Both tables will exist in the user account LAUREL. The first table was created by LAUREL, but the second table, TVSHOWS, was created by user account HARDY and was created as a table that is owned by LAUREL. The user account HARDY will contain no tables. The official "owner" of both tables is LAUREL, as the data dictionary confirms.

```
SELECT OWNER, TABLE_NAME
FROM   DBA_TABLES
WHERE  OWNER IN ('HARDY','LAUREL');

OWNER                          TABLE_NAME
------------------------------ ------------------------------
LAUREL                         MOVIES
LAUREL                         TVSHOWS
```

When a system privilege includes the keyword ANY in its title, it means that the privilege will authorize a user to perform the task as though they were any user account. In this example, user HARDY was able to create a table and place it in the LAUREL account, a task typically reserved only for user LAUREL. However, since user HARDY has the system privilege CREATE ANY TABLE, then HARDY can create any table in any user account.

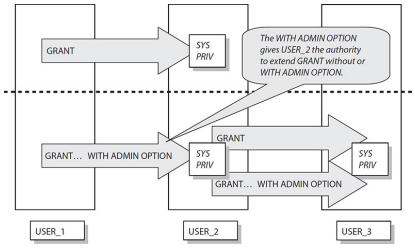
ADMIN OPTION

In a previous section we said we would look at the *option* in the GRANT statement's syntax we examined. Here it is: the *option* is an additional clause that may be included with the GRANT statement, as follows:

```
GRANT privilege TO user WITH ADMIN OPTION;
```

When any system privilege is granted with the WITH ADMIN OPTION option, then the recipient receives the system privilege itself, along with the right to grant the system privilege to another user (see Figure 14-4).

FIGURE 14-4 GRANT versus GRANT WITH ADMIN OPTION



The REVOKE statement does not use the WITH ADMIN OPTION clause. Whenever a system privilege is revoked, the entire system privilege is revoked.

If a user—let's call it the first user—grants a system privilege to a second user WITH ADMIN OPTION and the second user uses the admin option to grant that same system privilege to a third user, then the third user retains the privilege until it is explicitly revoked from the third user. In other words, once the second user has granted the third user with the system privilege, it stays with the first user, even if the first user—or any other qualified user—revokes the system privilege from the second user. If that happens, the third user still has the system privilege. The only way the third user will lose the system privilege is if any qualified user revokes the system privilege explicitly from the third user with a REVOKE statement. In other words, the REVOKE statement for system privileges does not "cascade." It applies only to the user to whom the revocation is applied.

ALL PRIVILEGES

As an alternative to granting specific system privileges, a qualified user account, such as SYSTEM or some other DBA qualified account, can issue the following statement:

```
GRANT ALL PRIVILEGES TO user;
```

This statement has the effect of granting all system privileges to the user. The WITH ADMIN OPTION clause may be used with this as well.

Needless to say, this should be done with great caution, if at all. It is not easily reversible. In other words, the following is not an exact counterpart:

```
REVOKE ALL PRIVILEGES FROM user;
```

This statement will reverse all system privileges granted to the user,

PUBLIC

The PUBLIC account is a built-in user account in the Oracle database that represents all users. Any objects owned by PUBLIC are treated as though they are owned by all the users in the database, present and future.

The GRANT statement will work with the keyword PUBLIC in the place of a user account name. Here's an example:

```
GRANT CREATE ANY TABLE TO PUBLIC;
```

This statement grants the CREATE ANY TABLE privilege to every user in the database. The CREATE ANY TABLE privilege gives every user the ability to create any table in any other user account. In other words, it's mass hysteria—or something like it. Mind you, we're not recommending you do this, but it's syntactically possible, and you need to be aware of it. While this sort of an example is unlikely, granting to PUBLIC may be useful with a selected number of object privileges, which we'll discuss a bit later.

Note that if you come to your senses and decide to revoke a system privilege from PUBLIC, you can do so without revoking any other system privileges. In other words, consider this statement:

```
REVOKE CREATE ANY TABLE FROM PUBLIC;
```

exam Watch

Note that if you want to grant all privileges, you use the keywords ALL PRIVILEGES. But if you want to grant certain privileges to all users, you do not use the keyword ALL. Instead, you grant to PUBLIC.

This statement will reverse the GRANT ... TO PUBLIC that we issued a few paragraphs earlier and thankfully will not revoke any individually granted CREATE ANY TABLE system privileges held by any user accounts. It will revoke only the GRANT to PUBLIC.

If you're even thinking about using GRANT ALL PRIVILEGES TO PUBLIC WITH ADMIN OPTION, you can put that thought out of your mind right this second.

CERTIFICATION OBJECTIVE 14.02

Grant Privileges on Tables and on a User

Any user with the system privilege CREATE TABLE can create a table. The table, once created, is owned by the user who created it. The owner does not require any explicitly granted privileges on the table. The table owner can use DML to add rows, change data in the table, query the data in the table, and remove rows from the table. But other users do not have that privilege automatically. Other users must have explicitly granted privileges on the object, which, in this case, is a table.

(Note: The exception, of course, is those users who have the system privileges that allow them to run any DML statements on any table in the database, regardless of who owns it. Those system privileges, as we saw in the previous section, include SELECT ANY TABLE, INSERT ANY TABLE, UPDATE ANY TABLE, and DELETE ANY TABLE.)

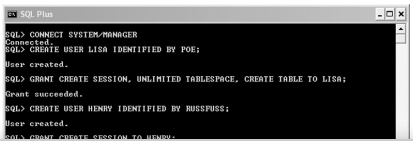
Any user who owns a table—or any other database object—may grant object privileges on their database object to other users in the database.

Object privileges exist for all DML statements—SELECT, INSERT, UPDATE, and DELETE—as well as any DDL statement that is relevant to an existing object, such as ALTER, for example. Note that there is no separate set of object privileges for the MERGE statement.

Object privileges on a table include all the DML statements that can be executed against a table. For example, if a user account LISA has the system privilege CREATE TABLE, then LISA can create a table. If LISA takes advantage of this system privilege and creates a table WEBINARS, then LISA can access the new table, but other users are not automatically able to see the table (unless, as we stated earlier, those users possess one of the ANY system privileges, such as SELECT ANY TABLE). To ensure that other user accounts can execute SQL statements on the table, user account LISA will have to grant object privileges on WEBINARS to other users.

See Figure 14-5 for a SQL*Plus session in which we connect to the SYSTEM account, where we create two user accounts, LISA and HENRY. We give LISA sufficient privileges to connect (CREATE SESSION) and create tables. Note how we combined multiple system privileges in a single GRANT statement. Also, we give HENRY sufficient privileges to create a session—but nothing more.

FIGURE 14-5 Creating, granting, and testing object privileges—part 1



We continue in Figure 14-6, where we connect to the LISA account and create a table, add data to it, and then grant privileges on the table to HENRY. Then we connect to HENRY, where we can issue SELECT and UPDATE statements but not INSERT—that particular privilege wasn't granted to HENRY.

FIGURE 14-6 Creating, granting, and testing object privileges—part 2

```
SQL Plus
SQL> CONNECT LISA/POE
Connected.
SQL> CREATE TABLE WEBINARS (WEBINAR_NAME VARCHAR2(20));

Table created.

SQL> INSERT INTO WEBINARS VALUES ('ONLINE DEMO');

1 row created.

SQL> GRANT SELECT, UPDATE ON WEBINARS TO HENRY;

Grant succeeded.

SQL> CONNECT HENRY/RUSSFUSS
Connected.
SQL> SELECT * FROM LISA.WEBINARS;

WEBINAR_NAME
--------------------
ONLINE DEMO

SQL> UPDATE LISA.WEBINARS SET WEBINAR_NAME = 'ONLINE TEST';

1 row updated.

SQL> INSERT INTO LISA.WEBINARS VALUES ('NEW ENTRY');
INSERT INTO LISA.WEBINARS VALUES ('NEW ENTRY')
            *
ERROR at line 1:
ORA-01031: insufficient privileges

SQL>
```

on the job

Take another look at Figure 14-6, and note the moment that the GRANT statement is issued. Remember that any DDL statement carries with it an implicit commit event. In other words, the GRANT statement has the effect of making the results of the INSERT statement permanent in the database. Once that GRANT has executed, the option to roll back the INSERT statement with ROLLBACK is no longer available.

Schema Prefixes

Note in Figure 14-6 that when HENRY references a table owned by LISA, HENRY must use the schema prefix to make the reference. In other words, HENRY could not issue a SELECT statement like this:

```
SELECT * FROM WEBINARS;
```

Instead, HENRY uses this sort of reference:

```
SELECT * FROM LISA.WEBINARS;
```

A SYNONYM is an object in the database that is an alternative name for a database object. A PUBLIC SYNONYM is a SYNONYM that is owned by the PUBLIC user account, which is an automatically created user account that is maintained by the Oracle database. The PUBLIC user isn't intended to be an account into which you log in to get access. Instead, PUBLIC is a mechanism by which you can create globally owned objects. Specifically, anything that is owned by PUBLIC is automatically owned by all users in the database. The same is true for PUBLIC SYNONYMS.

In our earlier example, the user SYSTEM could have given user LISA the system privilege to create public synonyms by issuing the following statement:

```
GRANT CREATE PUBLIC SYNONYM TO LISA;
```

Then, later, the user LISA could have used that system privilege to create a PUBLIC SYNONYM like this:

```
CREATE PUBLIC SYNONYM WEBINARS FOR LISA.WEBINARS;
```

Finally, once user HENRY got around to issuing DML statements on the WEBINARS table, HENRY could have omitted the schema prefix and instead simply executed this statement:

```
SELECT * FROM WEBINARS;
```

In this instance, HENRY would be specifying the WEBINARS object PUBLIC SYNONYM, which in turn points to the object LISA.WEBINARS. Note that no object privilege had to be granted on the PUBLIC SYNONYM object to HENRY. All objects owned by PUBLIC are automatically available and accessible to all users in the database, present and future. However, privileges must be granted to whatever object for which the PUBLIC SYNONYM serves as an alias. It's one thing to have privileges on a PUBLIC SYNONYM that references a table, but it's another thing to have privileges on the table it references. All users have privileges automatically on any object owned by PUBLIC; they do not have automatically granted privileges on anything a PUBLIC SYNONYM references; such privileges must be granted explicitly.

This sort of usage is the most common purpose of the PUBLIC SYNONYM object.

Note that to create PUBLIC SYNONYM objects, a user account must have the CREATE PUBLIC SYNONYM system privilege.

Name Priority, Revisited

You may recall our discussion in Chapter 2 about a concept called *namespace*. When a user makes a reference to an object by name, SQL will use that name to search for that object as follows:

and something called user-defined types (these objects are beyond the scope of the exam).

- Next, SQL looks in the database namespace, which contains users, roles, and public synonyms.

This concept was demonstrated graphically in Figure 2-2.

WITH GRANT OPTION

If you want to grant another user a particular object privilege and include the ability for the user to grant that same object privilege to yet another user, then include the WITH GRANT OPTION clause in the GRANT statement. Here's an example:

```
CONNECT LISA/POE
GRANT SELECT, UPDATE ON WEBINARS TO HENRY WITH GRANT OPTION;
```

This grant gives user HENRY the ability to issue SELECT and UPDATE statements on table WEBINARS, along with the ability to grant those privileges to other users. HENRY is not obligated to grant the set of privileges together; HENRY can choose to be selective.

```
CONNECT HENRY/RUSSFUSS
GRANT SELECT ON LISA.WEBINARS TO HAROLD WITH GRANT OPTION;
```

Now user HAROLD has the ability to issue SELECT statements on LISA.WEBINARS, as well as the ability to grant that privilege to others. But HENRY did not pass along the UPDATE privilege.

REVOKE

User LISA may choose to revoke privileges from HENRY, like this:

```
REVOKE SELECT, UPDATE ON WEBINARS FROM HENRY;
```

If user LISA does this, then HENRY and HAROLD lose all privileges, as does anyone to whom they extended privileges with their WITH GRANT OPTION option.

In other words, the revocation of object privileges "cascades." Note that this is different from system privilege revocation, which does not cascade, as we stated earlier in this chapter.

Note that the REVOKE statement does not require the WITH GRANT OPTION clause. REVOKE doesn't care whether that option had been included; it just revokes all specified privileges and cascades the change throughout all user accounts as required.

ALL PRIVILEGES

The ALL PRIVILEGES option works with granting and revoking object privileges in much the same way it does with system privileges, with some differences. Here's an example:

```
GRANT ALL PRIVILEGES ON WEBINARS TO HENRY;
```

This statement gives all privileges on the object WEBINARS to HENRY, except for the ability to grant privileges. To grant the ability to grant, use this:

```
GRANT ALL PRIVILEGES ON WEBINARS TO HENRY WITH GRANT OPTION;
```

The keyword PRIVILEGES is not required when granting object privileges.

```
GRANT ALL ON WEBINARS TO HENRY;
```

The same is true with REVOKE when used with object privileges.

```
REVOKE ALL PRIVILEGES ON WEBINARS FROM HENRY;
```

The following is also good:

```
REVOKE ALL ON WEBINARS FROM HENRY;
```

This shorthand way of revoking object privileges spares the effort of identifying all the individual object privileges that may have already been granted to HENRY on the WEBINARS table and revokes them all at once.

Note that the keyword PRIVILEGES is optional when working with object privileges, but not when working with system privileges.

If you use REVOKE ALL to revoke object privileges from a user and no object privileges exist on the object for that user, then no error message results, and the statement executes successfully with no practical effect.

Dependent Privileges

If user A owns a view, which is based on a table that user A also owns, and user A grants privileges on the view to user B, then user B can access the view without privileges to the underlying table.

If user A creates a table and a public synonym, then user B has immediate visibility of the public synonym because the synonym is owned by PUBLIC and all users have visibility of all objects owned by PUBLIC. However, user B still requires privileges on the table for which the public synonym is an alias. If the public synonym references a view that user A owns, then user B must have object privileges on the view, but is not required to have access to its underlying table.

If you grant privileges on a table and then drop the table, the privileges are dropped with the table. If you later re-create the table, you must also grant the privileges again. However, if you restore a dropped table with the FLASHBACK TABLE ... BEFORE DROP statement, you will recover the table, its associated indices, and the table's granted privileges, and you will not need to grant the privileges again.

View Privileges in the Data Dictionary

We've already looked at the data dictionary and seen how it provides information about the state of objects in the database, as well as providing some historic information.

There are many views in the data dictionary that present information about system privileges and object privileges. See Table 14-3 for a listing of some of these views.

TABLE 14-3 Data About Privileges in the Data Dictionary

Data Dictionary View	Explanation
USER_SYS_PRIVS	System privileges granted to current user
DBA_SYS_PRIVS	System privileges granted to users and roles
USER_TAB_PRIVS	Grants on objects for which the user is the grantor, grantee, or owner
ALL_TAB_PRIVS	Grants on objects for which the user is the grantor, grantee, owner, or an enabled role or PUBLIC is the grantee
DBA_TAB_PRIVS	Grants on all objects in the database
ALL_TAB_PRIVS_RECD	Grants on objects for which the user, PUBLIC, or enabled role is the grantee
SESSION_PRIVS	Privileges that are enabled to the user

For example, to see what system privileges are granted to your current user account, you can query the data dictionary view USER_SYS_PRIVS. Here's what the results might look like from user account LISA:

```
SELECT   PRIVILEGE, ADMIN_OPTION
FROM     USER_SYS_PRIVS
ORDER BY PRIVILEGE;

PRIVILEGE                                ADMIN_OPTION
---------------------------------------- ------------
CREATE PUBLIC SYNONYM                    NO
CREATE SESSION                           NO
CREATE TABLE                             NO
UNLIMITED TABLESPACE                     NO
```

The equivalent data dictionary view DBA_SYS_PRIVS allows you to see the same information for other users.

exam watch

A privilege may be granted directly as a privilege or indirectly as part of a role. If you intend to drop a privilege from a user, use the data dictionary to determine whether that same privilege is granted to a role that is also granted to the same user. If so, then the privilege you dropped directly is still granted to the user indirectly through the role.

To see all the object privileges that your current user account may have granted to others or that may have been granted by others, you can use the following query. This is what the results might look like within the user account LISA:

```
SELECT   GRANTOR, OWNER, GRANTEE, TABLE_NAME, PRIVILEGE, GRANTABLE
FROM     USER_TAB_PRIVS
ORDER BY GRANTOR, OWNER, GRANTEE, TABLE_NAME, PRIVILEGE;

GRANTOR    OWNER      GRANTEE    TABLE_NAME   PRIVILEGE    GRANTABLE
--------- ---------- ---------- ------------ ------------ ---------
EFCODD     EFCODD     LISA       PORTS        DELETE       NO
EFCODD     EFCODD     LISA       PORTS        INSERT       NO
EFCODD     EFCODD     LISA       PORTS        SELECT       NO
EFCODD     EFCODD     LISA       PORTS        UPDATE       NO
EFCODD     EFCODD     LISA       SHIPS        ALTER        NO
EFCODD     EFCODD     LISA       SHIPS        DELETE       NO
EFCODD     EFCODD     LISA       SHIPS        INSERT       NO
EFCODD     EFCODD     LISA       SHIPS        SELECT       NO
EFCODD     EFCODD     LISA       SHIPS        UPDATE       NO
LISA       LISA       HENRY      WEBINARS     SELECT       NO
LISA       LISA       HENRY      WEBINARS     UPDATE       NO
```

Note that the first several rows show object privileges granted by EFCODD to LISA. The final two rows show object privileges granted by LISA to HENRY.

These are just a few examples of the sort of information the data dictionary provides about system privileges and object privileges that have been granted to and from user accounts within the database.

on the job

When inspecting data dictionary views like DBA_TAB_PRIVS or DBA_SYS_PRIVS to see what privileges have been granted to a particular user account, you can check the GRANTEE column for the appropriate USER name. However, don't forget to also check for rows where GRANTEE = 'PUBLIC'; these privileges are also available to your user account.

Grant Roles

to a given role. Once it is created, you can grant a role to a user just as you can grant privileges to a user. The user is then automatically granted any privileges contained within the role. A role is an excellent way to manage the various privileges required for performing different tasks in the database and to organize the process of granting and revoking privileges.

You may grant the role to as many user accounts as you want.

If any privilege is subsequently revoked from the role, it is also revoked from any users to whom the role has been granted. In other words, changes to roles cascade to the users to whom the role is granted.

Three roles in particular have historically been associated with standard Oracle databases, but they are being phased out. On a practical level, though, it's good to know about them, if you don't already. The three roles are CONNECT, RESOURCE, and DBA. The CONNECT role consists of the CREATE SESSION system privilege, intended for the typical generic end user. RESOURCE is a collection of system privileges intended for the typical application developer. DBA is intended for the typical database administrator. Each can be seen in detail in the data dictionary view DBA_SYS_PRIVS (see Table 14-4 for details). All three roles are still included in each implementation of the Oracle database as of this writing, but Oracle has stated formally that the use of these roles is now officially discouraged, and their inclusion in future database implementations is not guaranteed. Oracle Corporation's official position is that you should create your own set of roles as required.

You can refer to the data in Table 14-4 to get an idea of the kind of system privileges you may want to include in your ROLE objects.

TABLE 14-4 The Classic Roles CONNECT, RESOURCE, and DBA

Role	Privilege
CONNECT	CREATE SESSION
RESOURCE	CREATE TRIGGER CREATE SEQUENCE CREATE TYPE CREATE PROCEDURE CREATE CLUSTER CREATE OPERATOR CREATE INDEXTYPE CREATE TABLE
DBA	More than 100 system privileges, including the following: CREATE ANY TABLE CREATE PUBLIC SYNONYM CREATE ROLE CREATE SYNONYM CREATE SEQUENCE CREATE USER CREATE VIEW GRANT ANY PRIVILEGE

To create a role, a user account needs the CREATE ROLE system privilege.

For example, the user account EFCODD owns several tables and wants to grant privileges on these tables to some users in the database. Some of these users will be performing queries on the tables and nothing more. Others will be responsible for performing changes to the data. Therefore, we want to create two different roles and grant them the necessary privileges.

```
CONNECT EFCODD/FOUNDER

CREATE ROLE CRUISE_ANALYST;
GRANT SELECT ON SHIPS     TO CRUISE_ANALYST;
GRANT SELECT ON PORTS     TO CRUISE_ANALYST;
GRANT SELECT ON EMPLOYEES TO CRUISE_ANALYST;

CREATE ROLE CRUISE_OPERATOR;

GRANT SELECT, UPDATE, INSERT, DELETE ON SHIPS     TO CRUISE_OPERATOR;
GRANT SELECT, UPDATE, INSERT, DELETE ON PORTS     TO CRUISE_OPERATOR;
GRANT SELECT, UPDATE                 ON EMPLOYEES TO CRUISE_OPERATOR;
```

In the preceding code, we create two ROLE objects: one called CRUISE_ANALYST, to which we grant some SELECT privileges on tables, and another called CRUISE_OPERATOR, to which we grant some other privileges. Once they are created, we can grant these roles to user accounts in the database.

```
GRANT CRUISE_OPERATOR TO LISA;
GRANT CRUISE_ANALYST  TO HENRY;
```

Once a role is granted, a user has access to all of the privileges within it.

A role can be granted to another role.

A role can be granted WITH ADMIN OPTION to empower the recipient to grant the role to yet another user. Here's an example:

```
GRANT CRUISE_OPERATOR TO LISA WITH ADMIN OPTION;
```

If a user grants a role to another user and uses WITH ADMIN OPTION, the second user may further grant the same role to a third user. If the first user revokes the role from the second user, the third user retains the role until it is explicitly revoked from the third user by a qualified user.

Table 14-5 lists some of the data dictionary views that provide information about existing roles in the database.

TABLE 14-5 Data Dictionary Views with Information About ROLE Objects

Data Dictionary View	Explanation
DBA_ROLES	All roles that exist in the database
DBA_ROLE_PRIVS	Roles granted to users and roles
DBA_SYS_PRIVS	System privileges granted to users and roles
DBA_TAB_PRIVS	All grants on objects to users and roles
ROLE_ROLE_PRIVS	Roles that are granted to roles
ROLE_SYS_PRIVS	System privileges granted to roles

Roles exist in a namespace that resides outside of any user account. Therefore, you can create roles with names that are the same as objects within a user account, such as tables and views. That's not necessarily a good idea, but it's allowed in the database.

A user account may be granted multiple roles at once.

Let's say you create an object, then grant a privilege on that object to a role, and then grant the role to a user. If you drop the object, then you also drop the granted object privilege to the role. However, the role still exists, and the grant of the role to the user still exists. If you subsequently re-create the object and then grant the object privilege to the role once again, then you've re-created the situation before the object was dropped. In other words, you do not need to re-create the role or grant the role to the user once again since neither was affected by the act of dropping the object on which the privilege had originally been granted.

CERTIFICATION OBJECTIVE 14.03

DISTINGUISH BETWEEN PRIVILEGES AND ROLES

A role object does not represent privileges in and of itself. It is merely a collection of zero or more privileges. A role exists independently of the privileges it may—or may not—contain. Furthermore, the relationship a user account has to a granted role is separate from any privileges that may have been granted directly to the user account. In other words, if a user account already has any object privileges granted directly to it as a result of earlier GRANT statements and then later is granted a role that duplicates any of those privileges, then the role exists separately from those originally granted privileges, which exist independently of the role. If the role is later revoked, that revocation does not adversely affect any separately granted privileges given directly to the user account.

If user HENRY were already granted a privilege that happens to be duplicated within the role CRUISE_ANALYST and then subsequently the role is granted but then is revoked, like this:

```
REVOKE CRUISE_ANALYST FROM HENRY;
```

then any object privileges granted directly to HENRY still exist.

For example, examine the following code:

```
01   GRANT SELECT ON INVOICES TO HENRY;
02   CREATE ROLE CRUISE_ACCOUNTANT;
03   GRANT SELECT ON INVOICES TO CRUISE_ACCOUNTANT;
04   GRANT CRUISE_ACCOUNTANT TO HENRY;
05   REVOKE CRUISE_ACCOUNTANT FROM HENRY;
```

User HENRY still has SELECT on INVOICES because of line 1, in spite of lines 2 through 5.

Similarly, if the role is restored but the direct object privilege is revoked, HENRY still has access through the role. Here's an example:

```
01   GRANT SELECT ON INVOICES TO HENRY;
02   CREATE ROLE CRUISE_ACCOUNTANT;
03   GRANT SELECT ON INVOICES TO CRUISE_ACCOUNTANT;
04   GRANT CRUISE_ACCOUNTANT TO HENRY;
05   REVOKE SELECT ON INVOICES FROM HENRY;
```

Remember that "privileges" may refer to either system privileges or object privileges, which are very different. Roles consist of some combination of one or more system and/or object privileges and/or other roles.

HENRY still has privileges on INVOICES in spite of line 5. The reason is the CRUISE_ACCOUNTANT role, from lines 2 through 4.

However, if the object privilege revoked from HENRY in line 5 were also to be revoked from the CRUISE_ACCOUNTANT role, then the object privilege would be removed from HENRY altogether.

CERTIFICATION SUMMARY

A system privilege is the right to perform a task in the database, using a DDL, DCL, or DML statement on objects in general. The right to perform those tasks on a particular object in the database is an object privilege. Finally, a role combines privileges into a single object so that a combination of privileges can be managed as a group.

The SQL statements GRANT and REVOKE are used to issue system

the CREATE SESSION system privilege. Other privileges include CREATE PUBLIC SYNONYM and CREATE TABLE.

The ANY keyword in a system privilege indicates the ability to work with objects that are owned by any user account.

A user account, by default, has object privileges on the objects it owns. Object privileges are required for a user to be able to interact with objects it does not own.

Instead of granting privileges to a user, you may create a role, then grant privileges to a role, and then grant the role to one or more users. The advantage is that if you have multiple users, a role is much easier to change since you can grant or revoke privileges as desired after the role has been assigned to any number of users, and all of the users will automatically have the new privileges granted or revoked automatically.

The data dictionary provides information about system privileges, object privileges, and roles from the perspective of both the grantor and the grantee.

✓ TWO-MINUTE DRILL

Differentiate System Privileges from Object Privileges

- [ ] The right to use any given SQL statement and/or to generally perform a task in the database is a system privilege.
- [ ] The right to perform some task on a specific existing object in the database is an object privilege.
- [ ] Both system and object privileges are granted to and revoked from users in the database.
- [ ] System privileges may be granted WITH ADMIN OPTION, which provides the ability for the recipient to grant the same privilege to yet another user.
- [ ] When a system privilege is revoked, the revocation does not cascade, meaning that it is revoked only from the user from whom it is being revoked, not from other users to whom the revoked user may have extended the privilege.
- [ ] The ALL PRIVILEGES keywords can be used to grant or revoke all privileges to or from a user.

Grant Privileges on Tables and on a User

- [ ] Object privileges correspond to DML statements and to DDL statements that are relevant to existing objects.
- [ ] Object privileges may be granted WITH GRANT OPTION, which provides the ability for the recipient to grant the same privilege to yet another user.
- [ ] When an object privilege is revoked, the revocation cascades, meaning that it is revoked from the user from whom it is being revoked, as well as from other users to whom the revoked user may have extended the privilege.
- [ ] When a user has been granted access to an object, the object name will require a schema name prefix to be correctly identified.
- [ ] A PUBLIC SYNONYM can provide an alternative name for the schema-prefixed version of the granted object.
- [ ] The ALL PRIVILEGES keywords can be used to grant or revoke all privileges to or from a user.
- [ ] A variety of data dictionary views provide information about system and object privileges.
- [ ] Users may see privileges granted to them, or granted by them to others, by querying the data dictionary.
- [ ] A role is created with the CREATE ROLE statement.
- [ ] Roles may be granted WITH ADMIN OPTION, which provides the ability for the recipient to grant the same role to yet another user.
- [ ] Roles exist in a namespace outside of an individual user account.
- [ ] A role is a collection of privileges and other roles.
- [ ] A role may be granted to another role.

Distinguish Between Privileges and Roles

- [ ] A privilege granted directly to a user exists independently from a privilege granted to a role.
- [ ] If you revoke a privilege directly from a user who also has been granted a role containing the same privilege, the role remains unchanged, and the user still has privileges by way of the role.
- [ ] The same situation is true with regard to revoking privileges directly from roles; if you revoke a privilege from a role that a user already has through a direct grant, the direct grant stays in force.

SELF TEST

Differentiate System Privileges from Object Privileges

1 . Which of the following SQL statements will authorize the user account JESSE to create tables in each and every user account in the database?

A. GRANT CREATE ALL TABLE TO JESSE;

B. GRANT CREATE PUBLIC TABLE TO JESSE;

C. GRANT CREATE ANY TABLE TO JESSE;

D. GRANT CREATE TABLE TO JESSE WITH PUBLIC OPTION;

2 . You are logged in to user account FRED and have been tasked with granting privileges to the user account ETHEL. You execute the following SQL statements:

```
GRANT CREATE ANY TABLE TO ETHEL WITH ADMIN OPTION;
REVOKE CREATE ANY TABLE FROM ETHEL;
```

Assuming both statements execute successfully, what is the result?

A. ETHEL does not have the system privilege CREATE ANY TABLE or the right to grant the CREATE ANY TABLE system privilege to any other user.

B. ETHEL has the system privilege CREATE ANY TABLE because the WITH ADMIN OPTION clause wasn't included in the REVOKE statement.

C. ETHEL no longer has the system privilege CREATE ANY TABLE but still has the right to grant the CREATE ANY TABLE system privilege to any other user, since the WITH ADMIN OPTION clause was omitted from the REVOKE statement. However, ETHEL may not grant the CREATE ANY TABLE privilege to herself.

D. ETHEL no longer has the system privilege CREATE ANY TABLE but still has the right to grant the CREATE ANY TABLE system privilege to any other user since the WITH ADMIN OPTION clause was omitted. Furthermore, ETHEL may grant the CREATE ANY TABLE privilege to herself because of the WITH ADMIN OPTION clause.

3 . Which of the following is the system privilege that is required as a minimum to allow a user account to log in to the database?

A. CREATE ANY LOGIN

B. CREATE ANY SESSION

C. CREATE SESSION

D. CREATE TABLE

4 . Which of the following is the system privilege that empowers the grantee to create an index in his or her own user account but not in the accounts of others?

A. CREATE TABLE

B. CREATE ANY TABLE

C. CREATE INDEX

D. CREATE ANY INDEX

Grant Privileges on Tables and on a User

5 . Your user account owns a table BACK_ORDERS, and you want to grant privileges on the table to a user account named CARUSO, which already has the system privileges CREATE SESSION and UNLIMITED TABLESPACE. Examine the following SQL statement:

```
GRANT SELECT ON BACK_ORDERS TO CARUSO;
```

Once this statement has been executed, which of the following statements will be true for user CARUSO?

A. CARUSO will have SELECT privileges on BACK_ORDERS but not the ability to give other users SELECT privileges on BACK_ORDERS.

B. CARUSO will have SELECT privileges on BACK_ORDERS, as well as the ability to give other users SELECT privileges on BACK_ORDERS.

C. CARUSO will have SELECT, INSERT, UPDATE, and DELETE privileges on BACK_ORDERS but not the ability to give other users those same privileges on BACK_ORDERS.

D. CARUSO will have SELECT and ALTER TABLE privileges on BACK_ORDERS but not the ability to give other users those same privileges on BACK_ORDERS.

6 . Your user account owns an updatable view, BACKLOG, which is based on the table PROJECTS. You are tasked to give SELECT and UPDATE capabilities to another user account named MARINO. Currently, MARINO has no privileges on either the table or the view. You want for MARINO to have the ability to grant SELECT on the view to other users as well. Examine the following SQL code:

```
GRANT SELECT ON BACKLOG TO MARINO WITH GRANT OPTION;
GRANT UPDATE ON BACKLOG TO MARINO;
```

A. The statements will fail, and MARINO will not be able to use the view.

B. The statements will execute successfully, but MARINO will not be able to SELECT from the view because the PROJECTS table has not been granted to MARINO.

C. The statements will execute successfully, and MARINO will be able to SELECT from the view but not UPDATE the view.

D. The statements will execute successfully and perform as intended.

7 . User account MUSKIE owns a table called CBAY. Which of the following statements can be executed by MUSKIE and enable user ONEILL to execute UPDATE statements on the CBAY table? (Choose three.)

A. GRANT ALL ON CBAY TO ONEILL;

B. GRANT ALL PRIVILEGES TO ONEILL;

C. GRANT ALL TO ONEILL;

D. GRANT INSERT, UPDATE ON CBAY TO ONEILL;

8 . Examine the following two claims:

[1] The DBA_TAB_PRIVS data dictionary view allows a user account to see object privileges it has granted to other user accounts.

[2] The DBA_TAB_PRIVS data dictionary view allows a user account to see object privileges granted by other user accounts to itself.

Which of these claims is true?

A. Only 1

B. Only 2

C. Both 1 and 2

D. Neither 1 nor 2

9 . Which of the following data dictionary views contains information about grants on tables that have been made by other users to your user account, as well as grants on tables that have been made by your user account to other user accounts?

A. USER_TAB_COLUMNS

B. USER_TAB_PRIVS

C. USER_TABLES

D. ALL_TAB_PRIVS_RECD

10. What can be granted to a role? (Choose all that apply.)

A. System privileges

B. Object privileges

C. Roles

D. None of the above

11. Which of the following statements will grant the role OMBUDSMAN to user JOSHUA in such a way that JOSHUA may grant the role to another user?

A. GRANT OMBUDSMAN TO JOSHUA WITH ADMIN OPTION;

B. GRANT OMBUDSMAN TO JOSHUA WITH GRANT OPTION;

C. GRANT OMBUDSMAN TO JOSHUA WITH ROLE OPTION;

D. GRANT OMBUDSMAN TO JOSHUA CASCADE;

12. User HARDING owns a table TEAPOT. User HARDING then executes the following SQL statements to give access to the table to user ALBERT:

```
CREATE PUBLIC SYNONYM TEAPOT FOR HARDING.TEAPOT;
CREATE ROLE DOME;
GRANT DOME TO ALBERT;
GRANT SELECT ON TEAPOT TO DOME;
```

Which of the following statements can user ALBERT now execute on the TEAPOT table?

A. SELECT * FROM DOME.HARDING.TEAPOT;

B. SELECT * FROM HARDING.DOME.TEAPOT;

C. SELECT * FROM HARDING.TEAPOT;

D. None of the above

Distinguish Between Privileges and Roles

13. A role:

A. Takes the place of privileges automatically so that any privilege granted to a role supersedes any grants that have already been granted directly to a user

D. Can be created by a user only if that user has the CREATE ROLE system privilege

14. You have a table FURNISHINGS and are told to grant DELETE privileges on the table to user HEARST. Examine the following SQL statements:

```
GRANT DELETE ON FURNISHINGS TO HEARST;
CREATE ROLE MGR;
GRANT DELETE ON FURNISHINGS TO MGR;
GRANT MGR TO HEARST;
```

Now you are told to change the privileges given to HEARST so that HEARST can no longer execute DELETE statements on the FURNISHINGS table. Which of the following will accomplish the goal? (Choose the best answer.)

A. REVOKE DELETE ON FURNISHINGS FROM HEARST;

B. REVOKE DELETE ON FURNISHINGS FROM MGR;

C. REVOKE DELETE ON FURNISHINGS FROM HEARST, MGR;

D. None of the above

15. Assume a database with three valid users: NEIL, BUZZ, and MICHAEL. Assume all users have the appropriate privileges they require to perform the tasks shown here. Assume NEIL owns a table called PROVISIONS. Examine the following code (assume all password references are valid):

```
01   CONNECT NEIL/neilPassword
02   GRANT SELECT ON PROVISIONS TO BUZZ, MICHAEL;
03
04   CONNECT BUZZ/buzzPassword
05   CREATE VIEW PROVISIONS AS SELECT * FROM NEIL.PROVISIONS;
06   GRANT SELECT ON PROVISIONS TO MICHAEL;
07   CREATE PUBLIC SYNONYM PROVISIONS FOR BUZZ.PROVISIONS;
08
09   CONNECT MICHAEL/michaelPassword
10   CREATE SYNONYM PROVISIONS FOR NEIL.PROVISIONS;
11   SELECT * FROM PROVISIONS;
```

What object is identified in line 11 by the name PROVISIONS?

A. The public synonym created in line 7

B. The synonym created in line 10

C. Nothing, because user NEIL did not include WITH GRANT OPTIONS in the GRANT SELECT ON PROVISIONS TO BUZZ statement

D. Something else not listed above

SELF TEST ANSWERS

Differentiate System Privileges from Object Privileges

1 . ☑ **C.** The system privilege CREATE ANY TABLE is the system privilege that you're looking for in this question. The keyword ANY is found in many system privileges to indicate that the user authorized with the system privilege may perform the task as though it were any user account in the database.
☒ **A, B,** and **D** are incorrect. There is no ALL keyword in this context, and PUBLIC does not apply here. There is no system privilege with the WITH PUBLIC OPTION keywords.

2 . ☑ **A.** The WITH ADMIN OPTION clause is not allowed nor needed in the REVOKE statement.
☒ **B, C,** and **D** are incorrect. They are all interesting ideas, but they are all wrong.

3 . ☑ **C.** The CREATE SESSION system privilege is the minimum requirement.
☒ **A, B,** and **D** are incorrect. There is no system privilege CREATE ANY LOGIN or CREATE ANY SESSION. CREATE TABLE is not required to establish a user session.

4 . ☑ **A.** The CREATE TABLE privilege also includes the ability to create an index. Remember that a CREATE TABLE statement may include the PRIMARY KEY or UNIQUE constraint, which—if created—will automatically cause the creation of an index to support each constraint.
☒ **B, C,** and **D** are incorrect. There isn't a CREATE INDEX system privilege. The ability is included with CREATE TABLE. CREATE ANY TABLE empowers the grantee the ability to create tables in the accounts of others, which potentially may also create indices in those same accounts. CREATE ANY INDEX is a valid system privilege for creating index objects in user accounts other than your own.

Grant Privileges on Tables and on a User

5 . ☑ **A.** GRANT SELECT ON *table* TO *user* gives the user the ability to SELECT on the table and nothing more.
☒ **B, C,** and **D** are incorrect. To give CARUSO the ability to SELECT on the table as well as to grant other users SELECT, the WITH GRANT OPTION clause would need to have been included with the GRANT statement, as in GRANT SELECT ON BACK_ORDERS TO CARUSO WITH GRANT OPTION. To grant the other DML statements on the table, each would have to have been included, as in GRANT SELECT, INSERT,

6 . ☑ **D.** The statements are syntactically correct and will perform as intended.
☒ **A, B,** and **C** are incorrect. The PROJECTS table does not need to be granted to MARINO since the VIEW has been granted. Since the VIEW is updatable, then the UPDATE privilege will work as well.

7 . ☑ **A, B,** and **D.** All three forms result in the UPDATE privilege being granted to user ONEILL for the CBAY table.
☒ **C** is incorrect. This statement is an invalid SQL statement. It either needs for the keyword PRIVILEGES to grant all system privileges to ONEILL or needs to name an object for which ALL privileges should be granted. The question is specifically asking about granting privileges on the CBAY table, so the ALL PRIVILEGES form would not work.

8 . ☑ **C.** The data dictionary view DBA_TAB_PRIVS allows a user to see privileges that have been granted to itself or by itself to others.
☒ **A, B,** and **D** are incorrect.

9 . ☑ **B.** USER_TAB_PRIVS is the correct answer.
☒ **A, C,** and **D** are incorrect. USER_TAB_COLUMNS has no information about grants. Neither does USER_TABLES. The ALL_TAB_PRIVS_RECD view contains data about incoming grants only.

10. ☑ **A, B,** and **C.** Both system and object privileges, as well as other roles, can be granted to any given role.
☒ **D** is incorrect.

11. ☑ **A.** WITH ADMIN OPTION is what is used for roles.
☒ **B, C,** and **D** are incorrect. WITH GRANT OPTION works for object privileges but not roles. There is no such clause as WITH ROLE OPTION. CASCADE does not apply here.

12. ☑ **C.** The schema name prefix correctly identifies the table. In addition, since the public synonym TEAPOT references the table, then DESC TEAPOT would also have worked—but that was not one of the options listed.
☒ **A, B,** and **D** are incorrect. You cannot use the role as a prefix or any other component of the name of a database object.

Distinguish Between Privileges and Roles

13. ☑ **D.** The CREATE ROLE privilege is required to create a role.
☒ **A, B,** and **C** are incorrect. A role does not replace privileges but instead is granted alongside of them. A role may be used to replace privileges as a management choice, and in fact such an approach is advisable, but it is not done automatically. Roles exist in a different namespace from tables and may duplicate table names. A user may be granted multiple roles at any given time.

14. ☑ **C.** This SQL statement accomplishes the goal in one statement.
☒ **A, B,** and **D** are incorrect. **A** and **B** are helpful but do not completely accomplish the task. **D** is incorrect because **C** is correct.

15. ☑ **B.** From within the MICHAEL user account, SQL first searches the local namespace and then searches the database namespace. The local namespace contains the private synonym, and that will be found first, before SQL looks in the database namespace.
☒ **A, C,** and **D** are incorrect. The GRANT statement issued by NEIL does not require WITH GRANT OPTION for the synonyms to function.

PREV
13 Manipulating Large Data Sets

NEXT
A About the Download