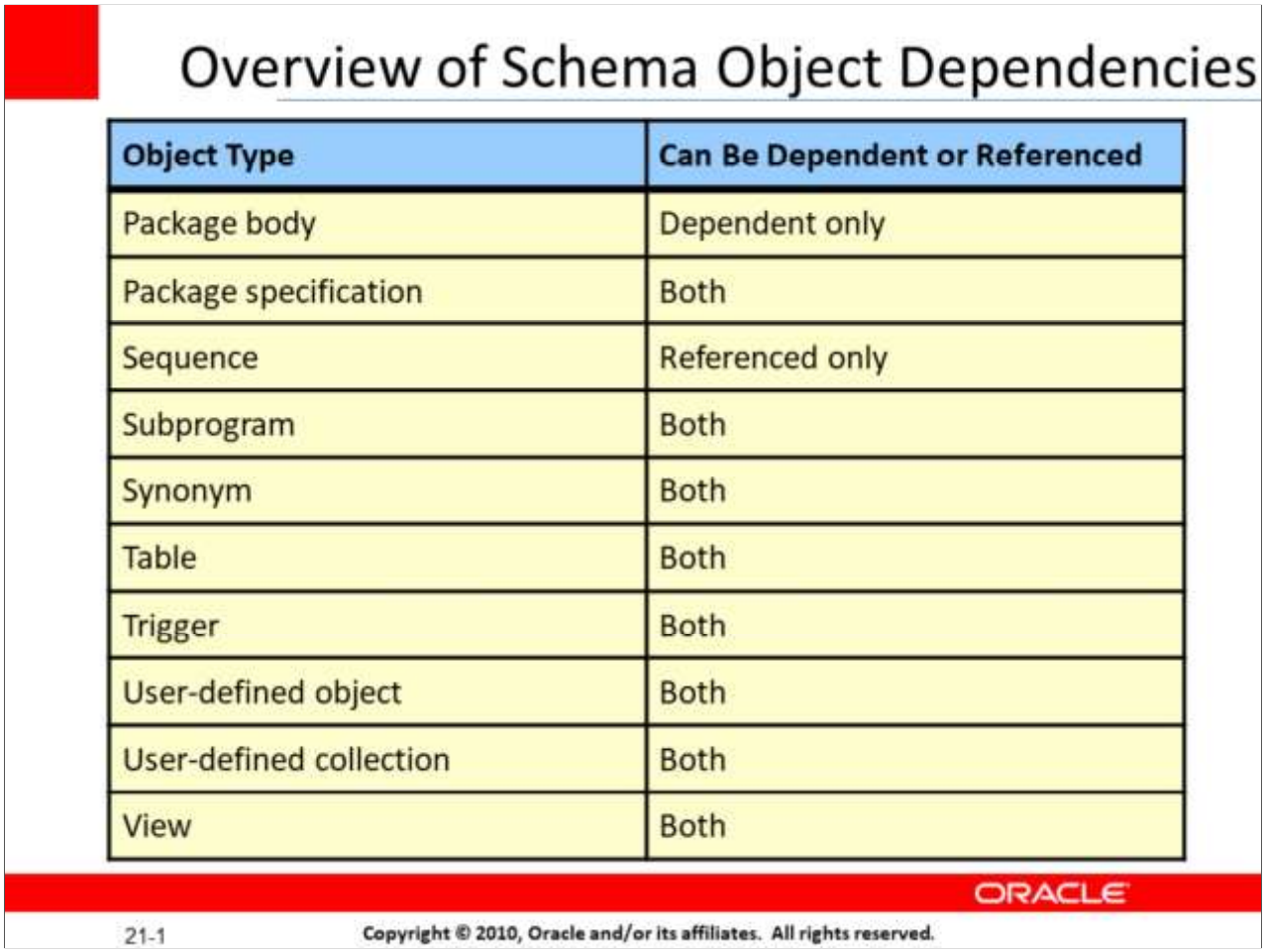
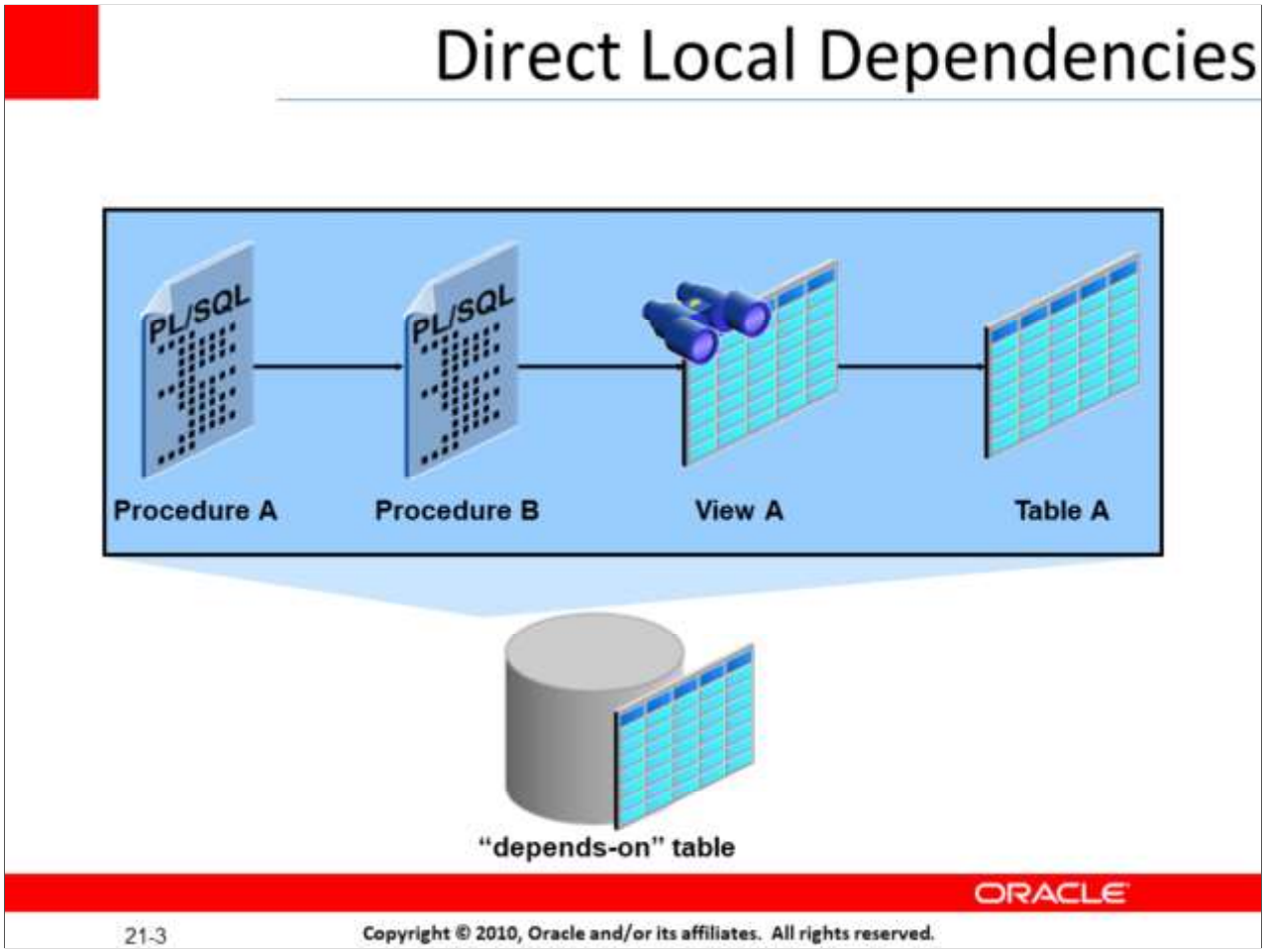


Dependent and Referenced Objects

Some types of schema objects can reference other objects in their definitions. For example, a view is defined by a query that references tables or other views, and the body of a subprogram can include SQL statements that reference other objects. If the definition of object A references object B, then A is a dependent object (with respect to B) and B is a referenced object (with respect to A).

Dependency Issues

- If you alter the definition of a referenced object, dependent objects may or may not continue to work properly. For example, if the table definition is changed, the procedure may or may not continue to work without error.
- The Oracle server automatically records dependencies among objects. To manage dependencies, all schema objects have a status (valid or invalid) that is recorded in the data dictionary, and you can view the status in the USER_OBJECTS data dictionary view.
- If the status of a schema object is VALID, then the object has been compiled and can be immediately used when referenced.
- If the status of a schema object is INVALID, then the schema object must be compiled before it can be used.



Dependent and Referenced Objects (continued)

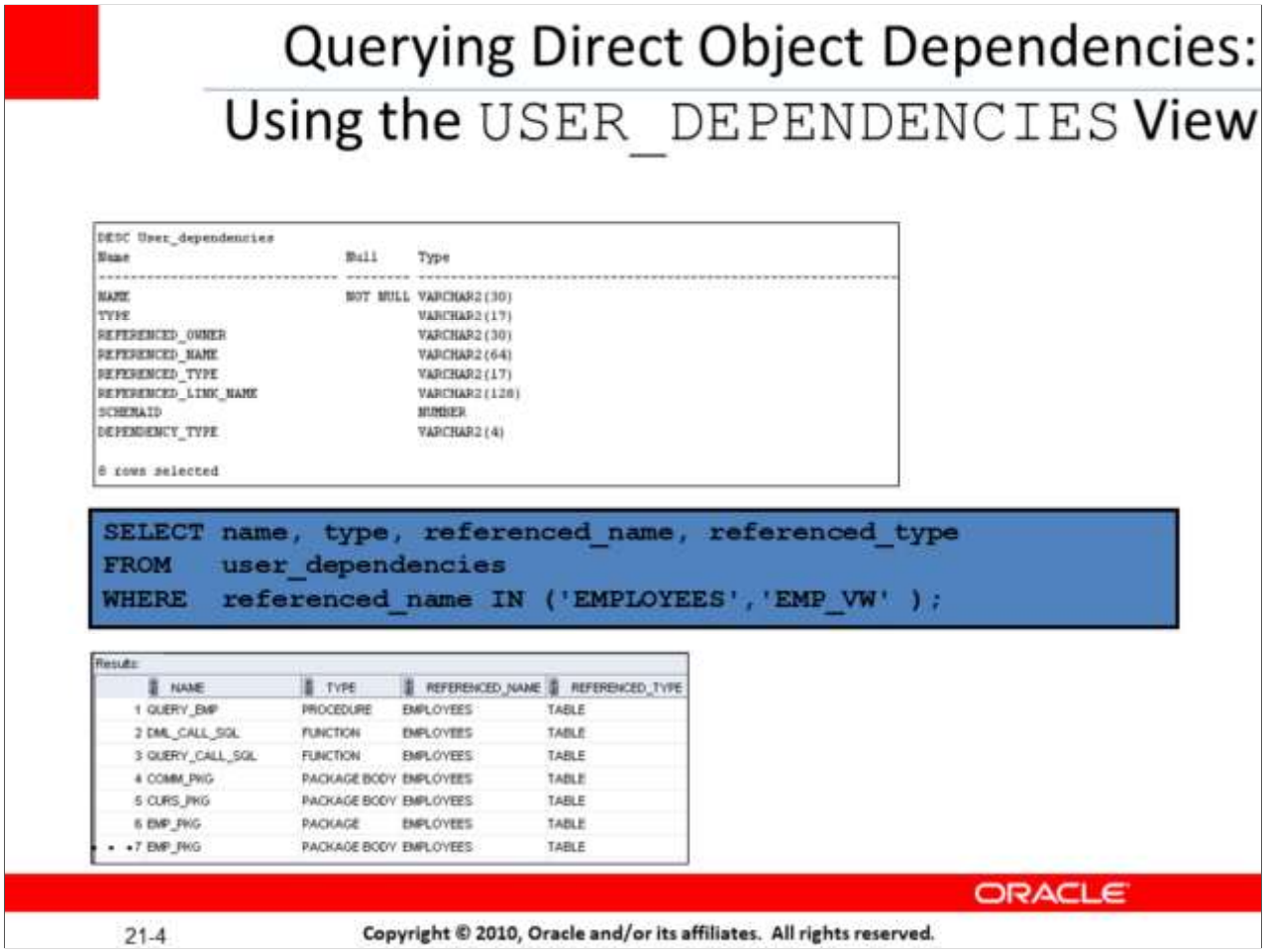
A procedure or function can directly or indirectly (through an intermediate view, procedure, function, or packaged procedure or function) reference the following objects:

- Tables
- Views
- Sequences
- Procedures
- Functions
- Packaged procedures or functions

Managing Local Dependencies

In the case of local dependencies, the objects are on the same node in the same database. The Oracle server automatically manages all local dependencies, using the database's internal "depends-on" table. When a referenced object is modified, the dependent objects are sometimes invalidated. The next time an invalidated object is called, the Oracle server automatically recompiles it.

If you alter the definition of a referenced object, dependent objects might or might not continue to function without error, depending on the type of alteration. For example, if you drop a table, no view based on the dropped table is usable.



Starting with Oracle Database 10g, the CREATE OR REPLACE SYNONYM command has been enhanced to minimize the invalidations to dependent PL/SQL program units and views that reference it. This is covered later in this lesson.

Starting with Oracle Database 11g, dependencies are tracked at the level of element within unit. This is referred to as fine-grained dependency. Fine-grained dependencies are covered later in this lesson.

Querying Direct Object Dependencies: Using the USER_DEPENDENCIES View

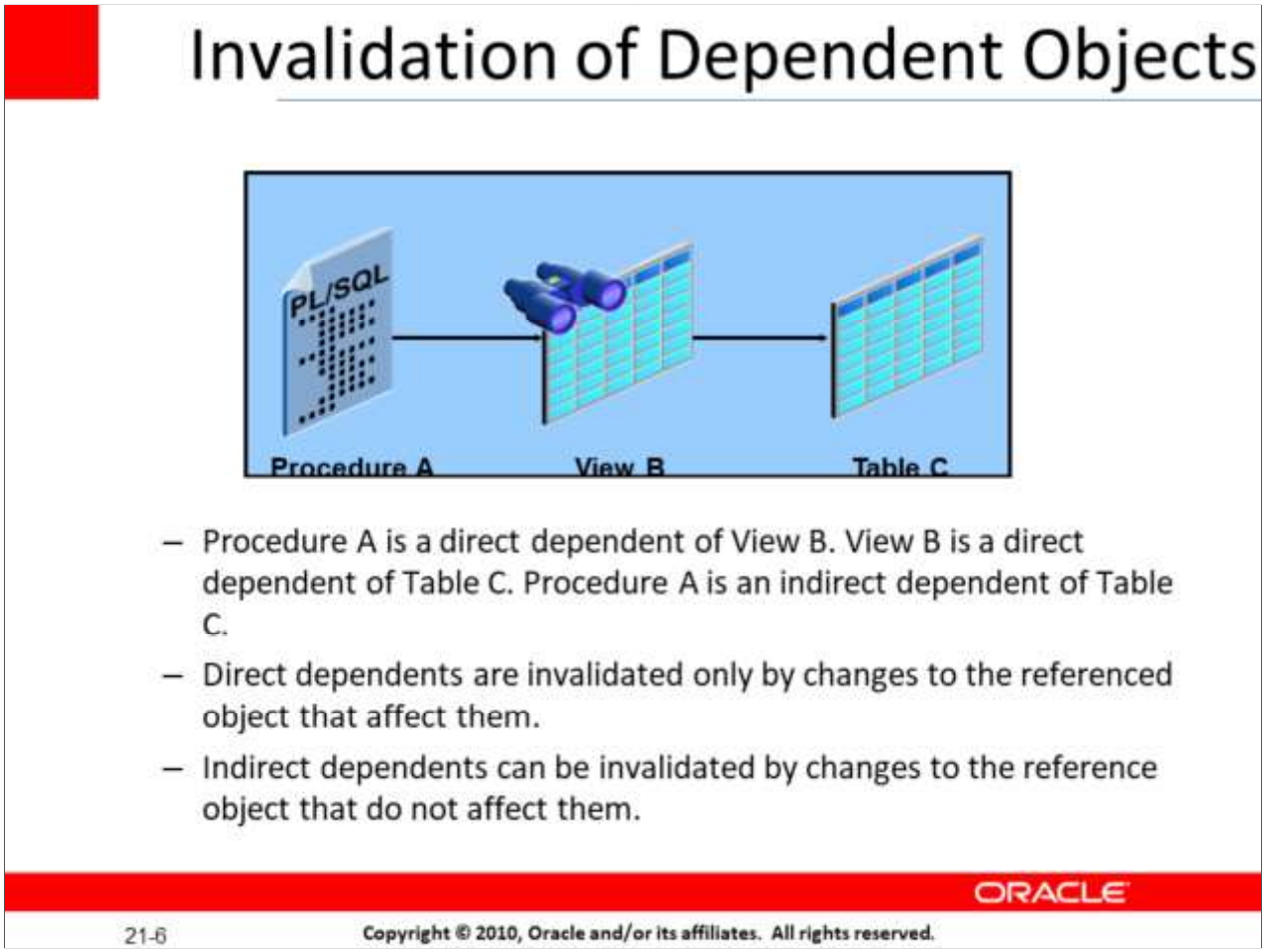
You can determine which database objects to recompile manually by displaying direct dependencies from the USER_DEPENDENCIES data dictionary view.

The ALL_DEPENDENCIES and DBA_DEPENDENCIES views contain the additional OWNER column, which references the owner of the object.

The USER_DEPENDENCIES Data Dictionary View Columns

The columns of the USER_DEPENDENCIES data dictionary view are as follows:

- NAME: The name of the dependent object
- TYPE: The type of the dependent object (PROCEDURE, FUNCTION, PACKAGE, PACKAGE BODY, TRIGGER, or VIEW)
- REFERENCED_OWNER: The schema of the referenced object



- REFERENCED_NAME: The name of the referenced object
- REFERENCED_TYPE: The type of the referenced object
- REFERENCED_LINK_NAME: The database link used to access the referenced object Querying an Object's Status

Every database object has one of the status values shown in the table in the slide.

Note: The USER_OBJECTS, ALL_OBJECTS, and DBA_OBJECTS static data dictionary views do not distinguish between Compiled with errors, Invalid, and Unauthorized; instead, they describe all these as INVALID.

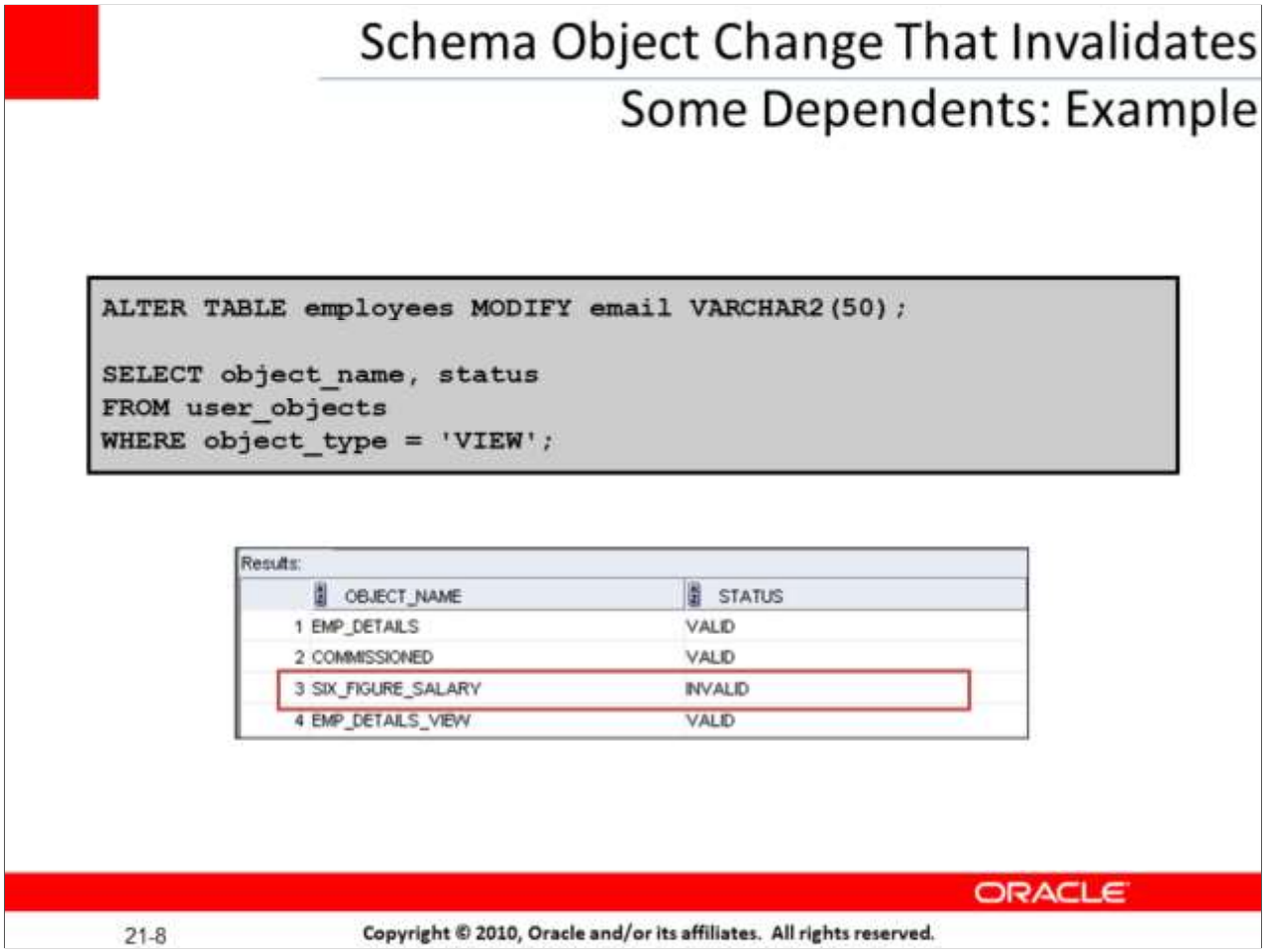
Invalidation of Dependent Objects

If object A depends on object B, which depends on object C, then A is a direct dependent of B, B is a direct dependent of C, and A is an indirect dependent of C.

In Oracle Database 11g, direct dependents are invalidated only by changes to the referenced object that affect them (changes to the signature of the referenced object).

Indirect dependents can be invalidated by changes to the reference object that do not affect them: If a change to Table C invalidates View B, it invalidates Procedure A (and all other direct and indirect dependents of View B). This is called cascading invalidation.

Assume that the structure of the table on which a view is based is modified. When you describe the view by using the SQL*Plus DESCRIBE command, you get an error message that states that



the object is invalid to describe. This is because the command is not a SQL command; at this stage, the view is invalid because the structure of its base table is changed. If you query the view now, then the view is recompiled automatically and you can see the result if it is successfully recompiled.

Schema Object Change That Invalidates Some Dependents: Example

The example in the slide demonstrates an example of a schema object change that invalidates some dependents but not others. The two newly created views are based on the EMPLOYEES table in the HR schema. The status of the newly created views is VALID.

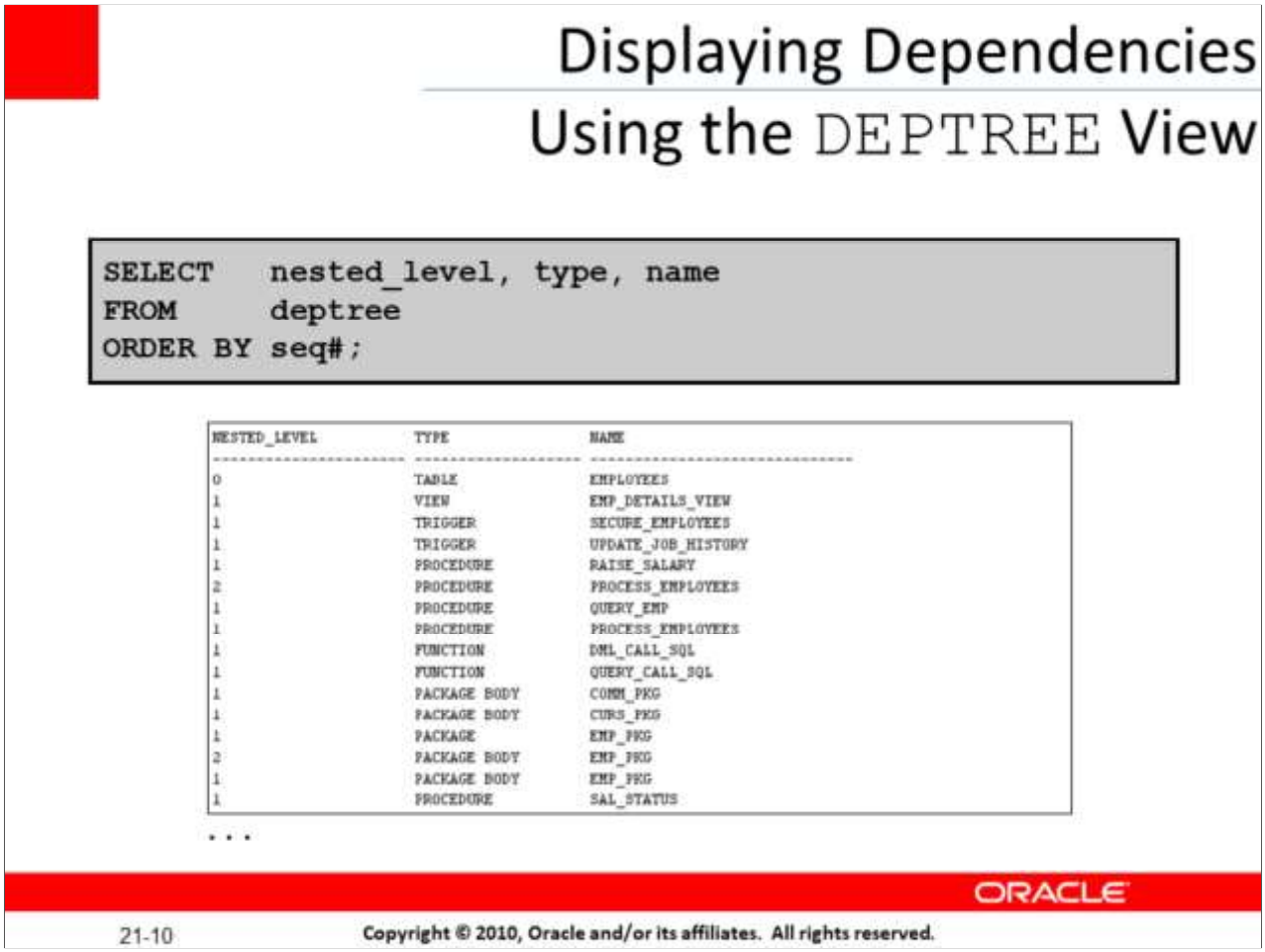
Schema Object Change That Invalidates Some Dependents: Example (continued)

Suppose you determine that the EMAIL column in the EMPLOYEES table needs to be lengthened from 25 to 50, you alter the table as shown in the slide above.

Because the COMMISSIONED view does not include the EMAIL column in its select list, it is not invalidated. However, the SIXFIGURES view is invalidated because all columns in the table are selected.

Displaying Direct and Indirect Dependencies by Using Views Provided by Oracle

Display direct and indirect dependencies from additional user views called DEPTREE and IDEPTREE; these views are provided by Oracle.



Example

1. Make sure that the utldtree.sql script has been executed. This script is located in the $ORACLE_HOME/labs/plpu/labs folder. You can run the script as follows:

   @?/labs/plpu/labs/utldtree.sql

   Note: In this class, this script is supplied in the labs folder of your class files. The code example above uses the student account ORA61. (This applies to a Linux environment. If the file is not found, locate the file in your labs subdirectory.)
2. Populate the DEPTREE_TEMPTAB table with information for a particular referenced object by invoking the DEPTREE_FILL procedure. There are three parameters for this procedure:

   object_type Type of the referenced object object_owner Schema

   of the referenced object object_name Name of the referenced object

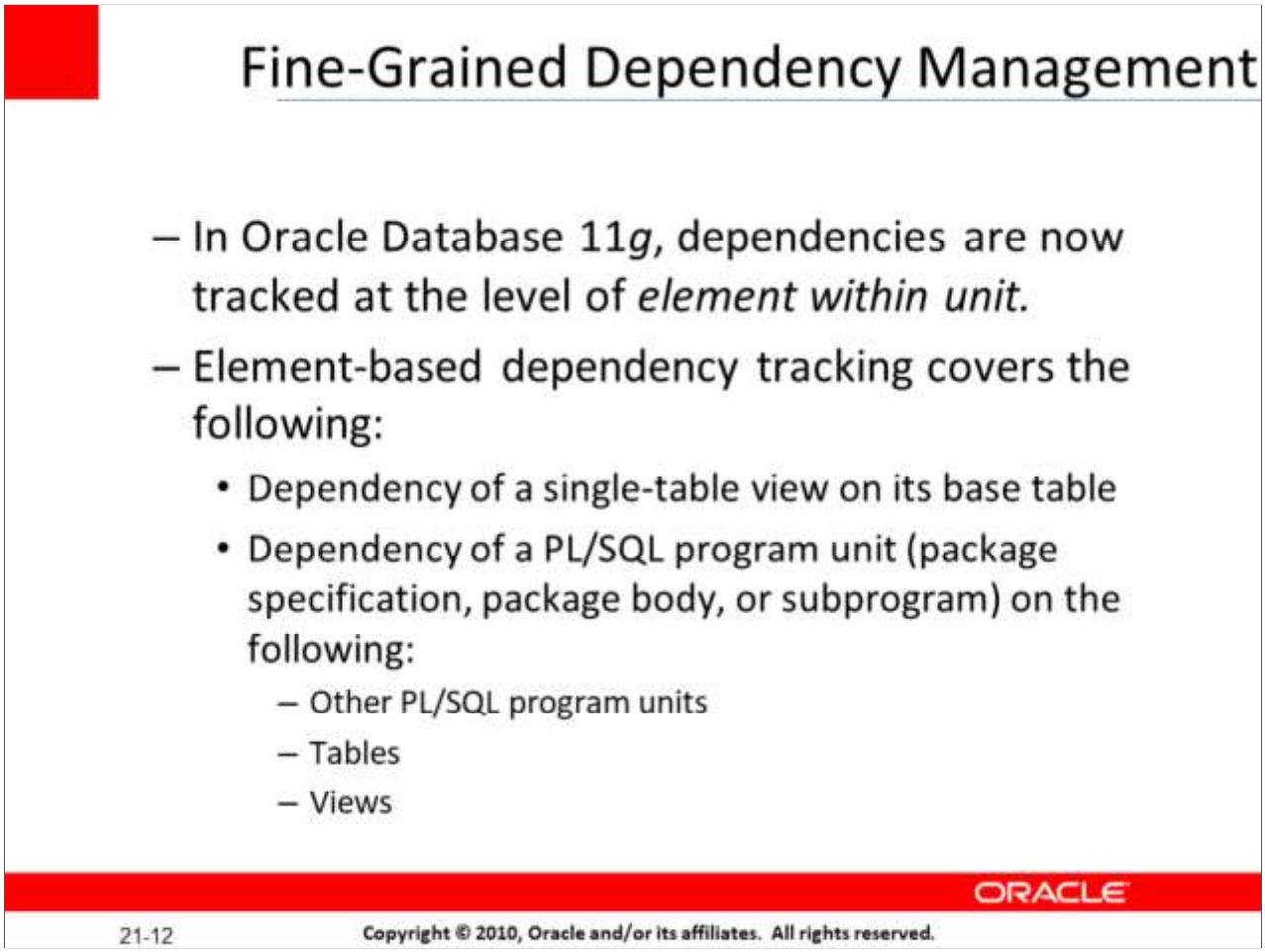
Displaying Dependencies Using the DEPTREE View

You can display a tabular representation of all dependent objects by querying the DEPTREE view. You can display an indented representation of the same information by querying the IDEPTREE view, which consists of a single column named DEPENDENCIES as follows:

```
SELECT  *
FROM    ideptree;
```

```
DEPENDENCIES
--------------------------------------------------------------------------------
   TRIGGER ORA61.SECURE_EMPLOYEES
   PROCEDURE ORA61.UPDATE_SALARY
   TRIGGER ORA61.AUDIT_EMP_VALUES
   PROCEDURE ORA61.EMP_LIST
   PROCEDURE ORA61.PROCESS_EMPLOYEES
   PACKAGE BODY ORA61.CURS_PKG
      PACKAGE BODY ORA61.EMP_PKG
   VIEW ORA61.EMP_DETAILS
      TRIGGER ORA61.NEW_EMP_DEPT
   TRIGGER ORA61.UPDATE_JOB_HISTORY
   PACKAGE BODY ORA61.EMP_PKG
   FUNCTION ORA61.EMP_HIRE_DATE
   PROCEDURE ORA61.SAL_STATUS
   TRIGGER ORA61.DERIVE_COMMISSION_PCT
   TRIGGER ORA61.CHECK_SALARY
      PROCEDURE ORA61.PROCESS_EMPLOYEES
   PROCEDURE ORA61.QUERY_EMP
   TRIGGER ORA61.RESTRICT_SALARY
   VIEW ORA61.COMMISSIONED
```

. . .

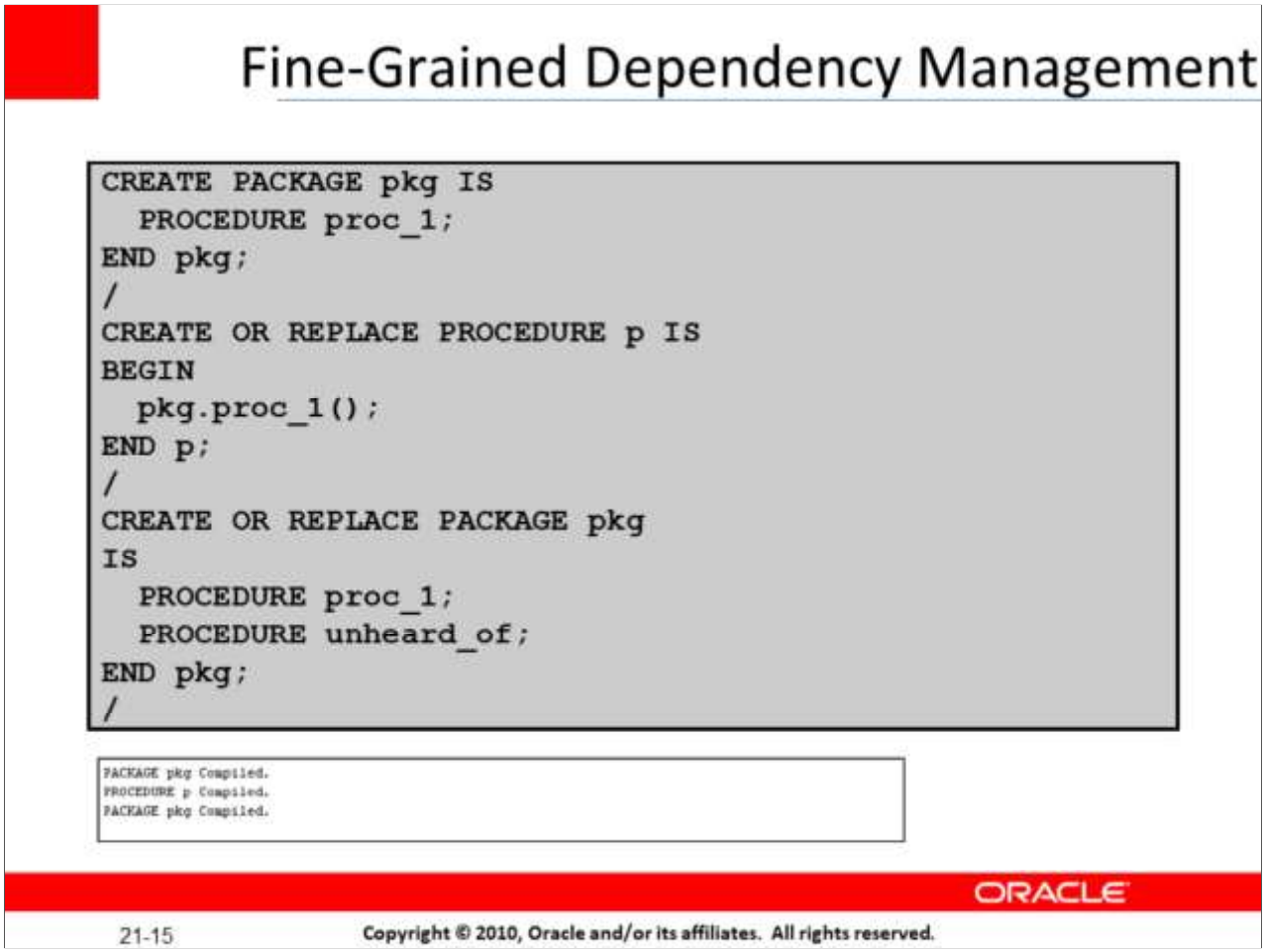


Fine-Grained Dependencies

Starting with Oracle Database 11g, you have access to records that describe more precise dependency metadata. This is called fine-grained dependency and it enables you to see when the dependent objects are not invalidated without logical requirement.

Earlier Oracle Database releases record dependency metadata—for example, PL/SQL unit P depends on PL/SQL unit F, or that view V depends on table T—with the precision of the whole object. This means that dependent objects are sometimes invalidated without logical requirement. For example, if view V depends only on columns A and B in table T, and column D is added to table T, the validity of view V is not logically affected. Nevertheless, before Oracle Database Release 11.1, view V is invalidated by the addition of column D to table T. With Oracle Database Release 11.1, adding column D to table T does not invalidate view V. Similarly, if procedure P depends only on elements E1 and E2 within a package, adding element E99 to the package does not invalidate procedure P.

Reducing the invalidation of dependent objects in response to changes to the objects on which they depend increases application availability, both in the development environment and during online application upgrade.



Fine-Grained Dependency Management: Example 1 Example of Dependency of a Single-Table View on Its Base Table

In the first example in the slide, table T2 is created with three columns: COL_A, COL_B, and COL_C. A view named V is created based on columns COL_A and COL_B of table T2. The dictionary views are queried and the view V is dependent on table T and its status is valid.

In the third example, table T2 is altered. A new column named COL_D is added. The dictionary views still report that the view V is dependent because element-based dependency tracking realizes that the columns COL_A and COL_B are not modified and, therefore, the view does not need to be invalidated.

Fine-Grained Dependency Management: Example 1 (continued)

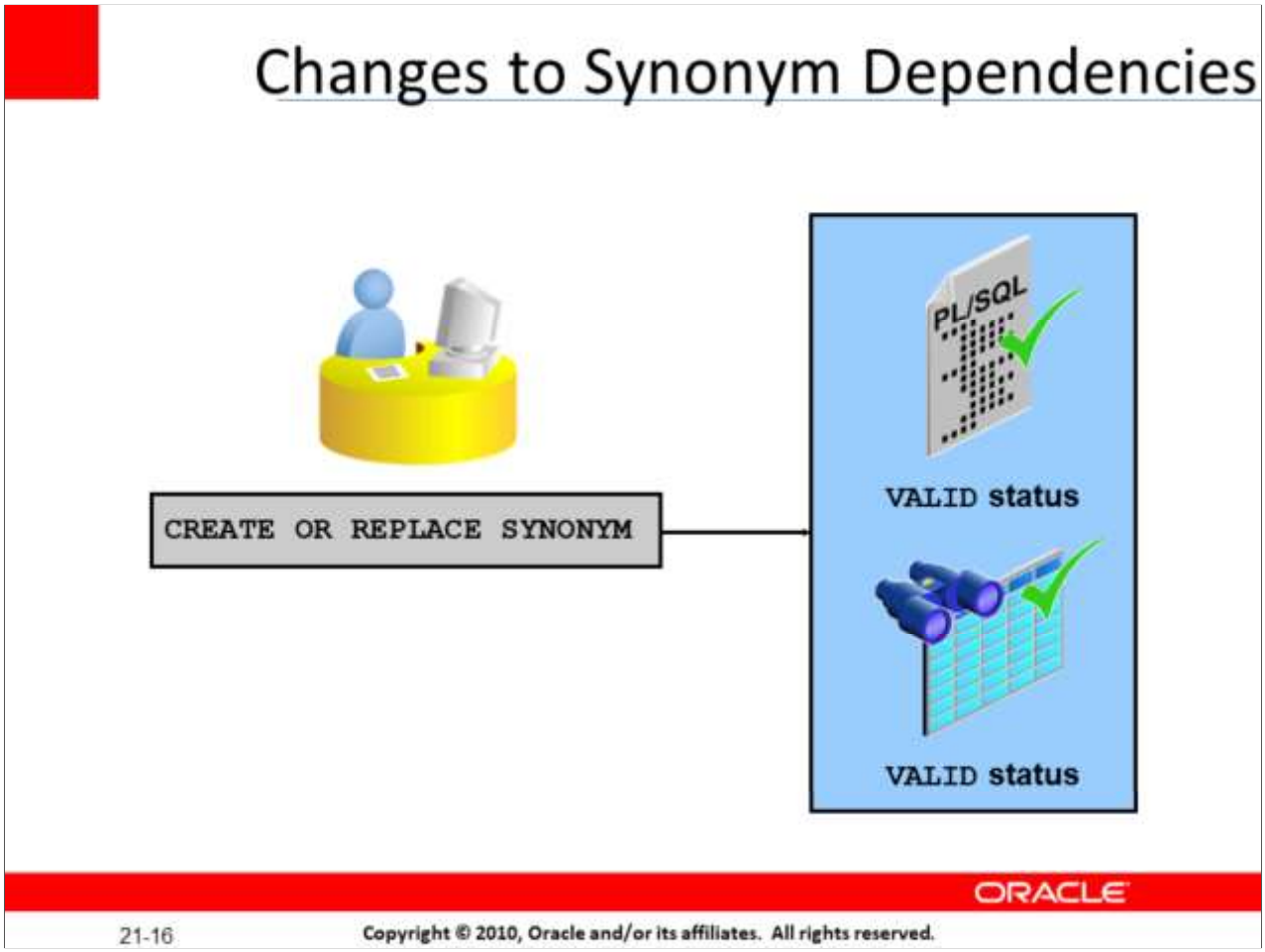
In the example in the slide, the view is invalidated because its element (COL_A) is modified in the table on which the view is dependent.

Fine-Grained Dependency Management: Example 2

In the example in the slide, you create a package named PKG that has procedure PROC_1 declared.

A procedure named P invokes PKG.PROC_1.

The definition of the PKG package is modified and another subroutine is added to the package declaration.



When you query the USER_OBJECTS dictionary view for the status of the P procedure, it is still valid as shown because the element you added to the definition of PKG is not referenced through procedure P.

```
SELECT status FROM user_objects
WHERE object_name = 'P';
```

Results:

	STATUS
1	VALID

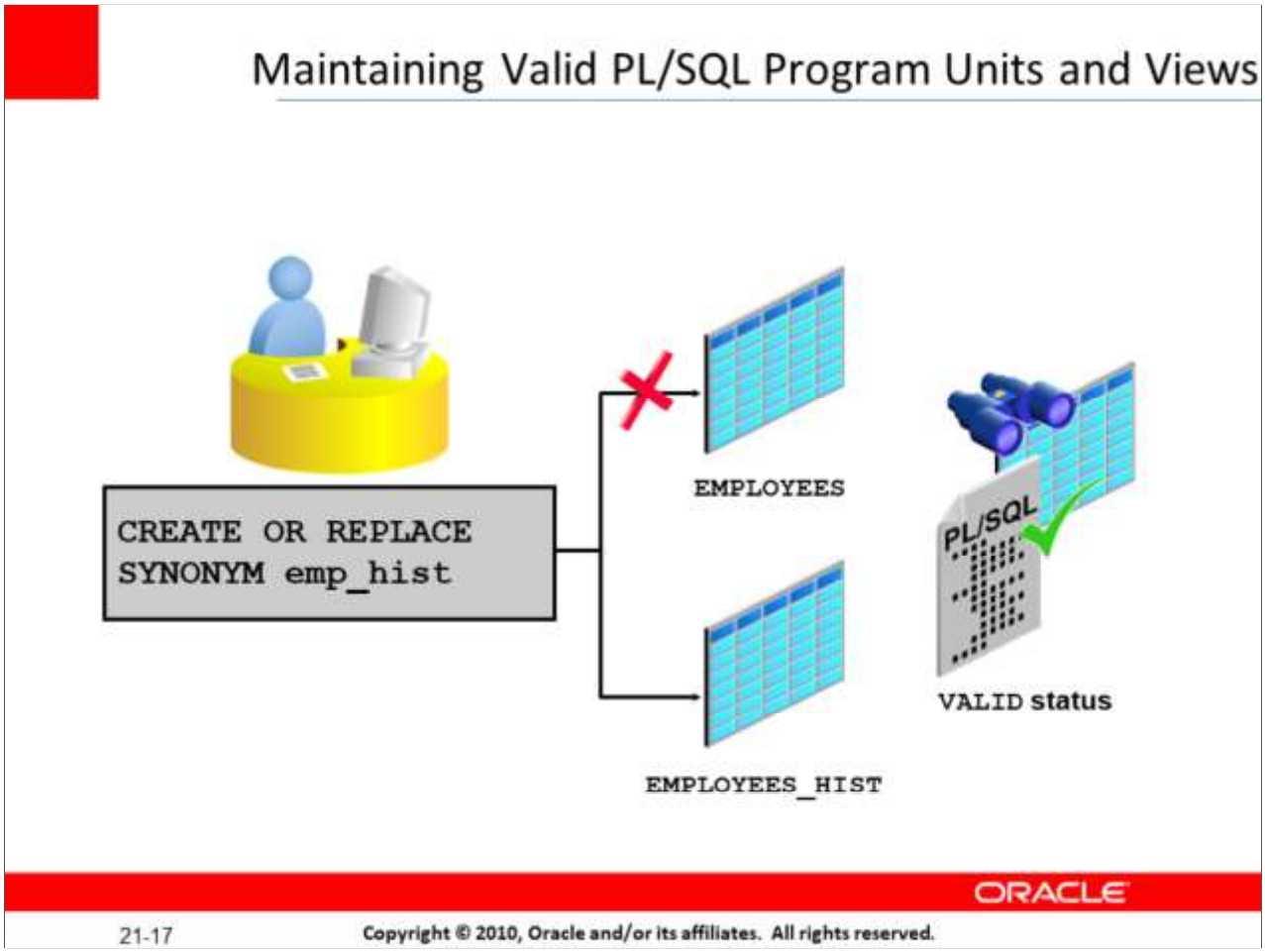
Changes to Synonym Dependencies

The Oracle Database minimizes down time during code upgrades or schema merges.

When certain conditions on columns, privileges, partitions, and so on are met, a table or object type is considered equivalent and dependent objects are no longer invalidated.

In Oracle Database 10g, the CREATE OR REPLACE SYNONYM command has been enhanced to minimize the invalidations to dependent PL/SQL program units and views that reference it. This eliminates the need for time-consuming recompilation of the program units after redefinition of the synonyms or during execution. You do not have to set any parameters or issue any special commands to enable this functionality; invalidations are minimized automatically.

Note: This enhancement applies only to synonyms pointing to tables.



Maintaining Valid PL/SQL Program Units

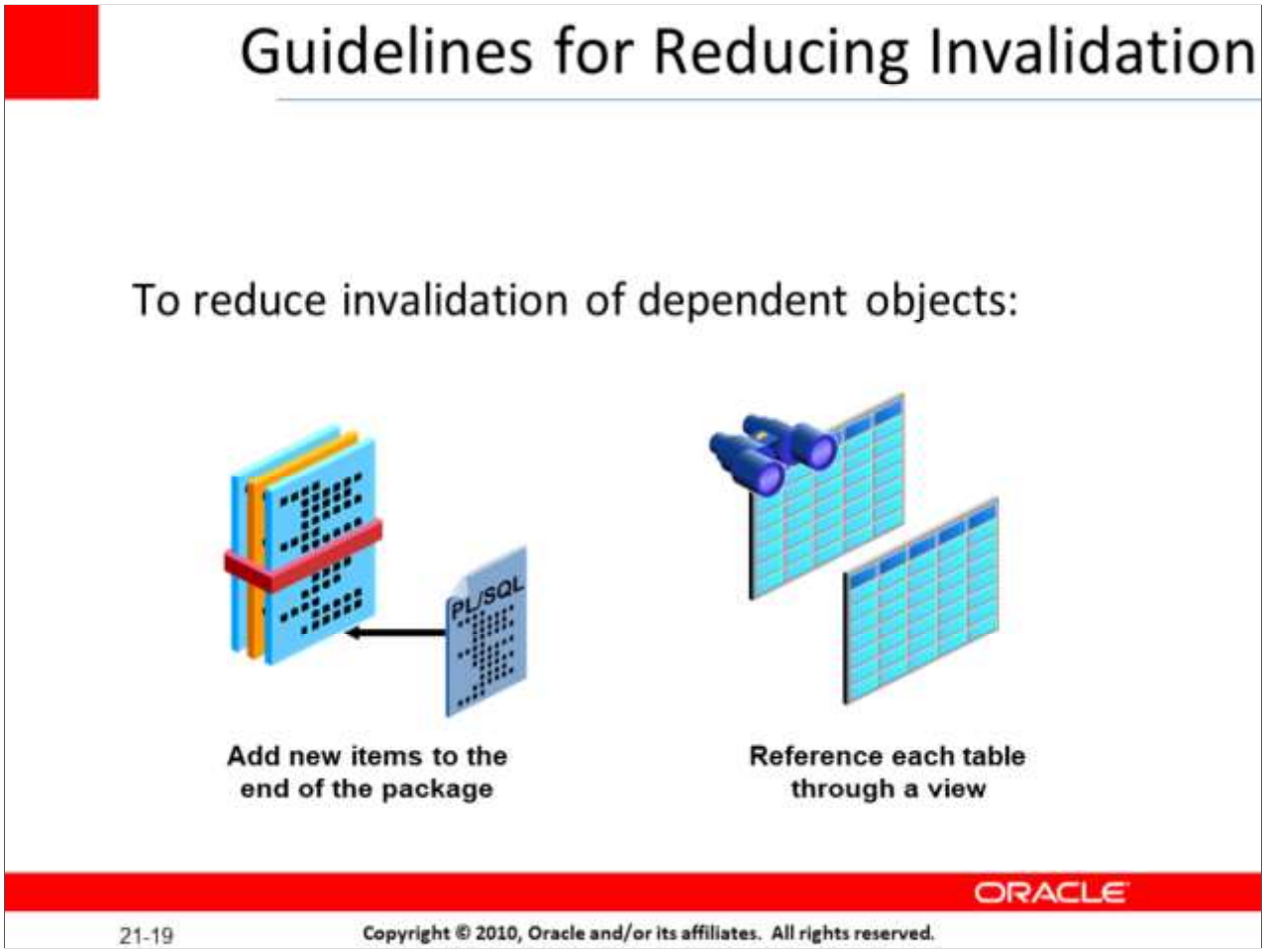
Starting with Oracle Database 10g Release 2, you can change the definition of a synonym, and the dependent PL/SQL program units are not invalidated under the following conditions:

- The column order, column names, and column data types of the tables are identical.
- The privileges on the newly referenced table and its columns are a superset of the set of privileges on the original table. These privileges must not be derived through roles alone.
- The names and types of partitions and subpartitions are identical.
- The tables are of the same organization type.
- Object type columns are of the same type.

Maintaining Valid Views

As with dependent PL/SQL program units, you can change the definition of a synonym, and the dependent views are not invalidated under the conditions listed in the preceding paragraph. In addition, the following must be true to preserve the VALID status of dependent views, but not of dependent PL/SQL program units, when you redefine a synonym:

- Columns and order of columns defined for primary key and unique indexes, NOT NULL constraints, and primary key and unique constraints must be identical.
- The dependent view cannot have any referential constraints.



Another Scenario of Local Dependencies

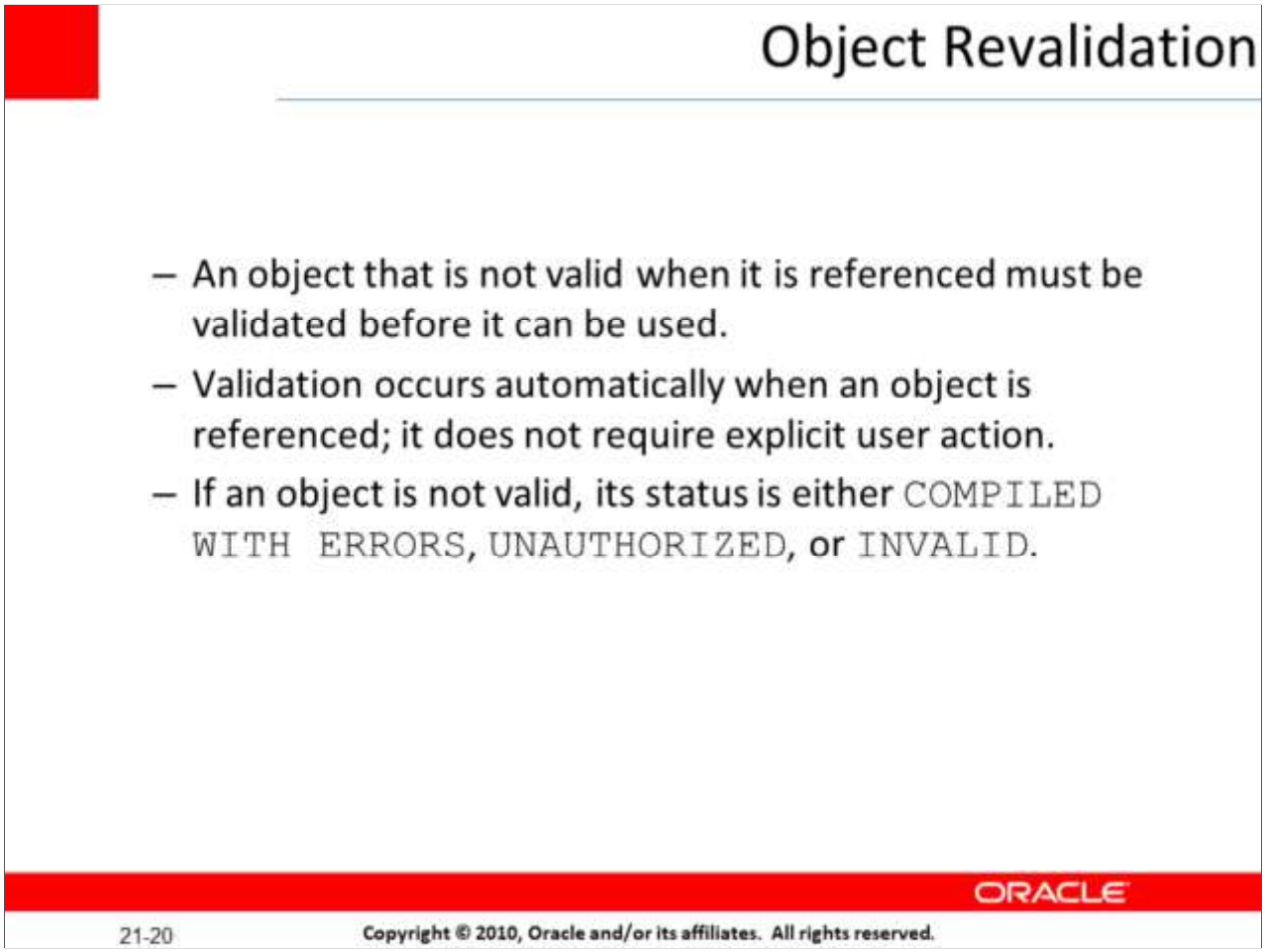
The example in the slide illustrates the effect that a change in the definition of a procedure has on the recompilation of a dependent procedure. Assume that the RAISE_SAL procedure updates the EMPLOYEES table directly, and that the REDUCE_SAL procedure updates the EMPLOYEES table indirectly by way of RAISE_SAL.

- If the internal logic of the RAISE_SAL procedure is modified, REDUCE_SAL will successfully recompile if RAISE_SAL has successfully compiled.
- If the formal parameters for the RAISE_SAL procedure are eliminated, REDUCE_SAL will not successfully recompile.

Guidelines for Reducing Invalidation Add New Items to End of Package

When adding new items to a package, add them to the end of the package. This preserves the slot numbers and entry-point numbers of existing top-level package items, preventing their invalidation. For example, consider the following package:

```
CREATE OR REPLACE PACKAGE pkg1 IS
FUNCTION get_var RETURN VARCHAR2;
PROCEDURE set_var (v VARCHAR2); END;
```



Adding an item to the end of pkg1 does not invalidate dependents that reference get_var. Inserting an item between the get_var function and the set_var procedure invalidates dependents that reference the set_var function.

Reference Each Table Through a View

Reference tables indirectly, using views. This allows you to do the following:

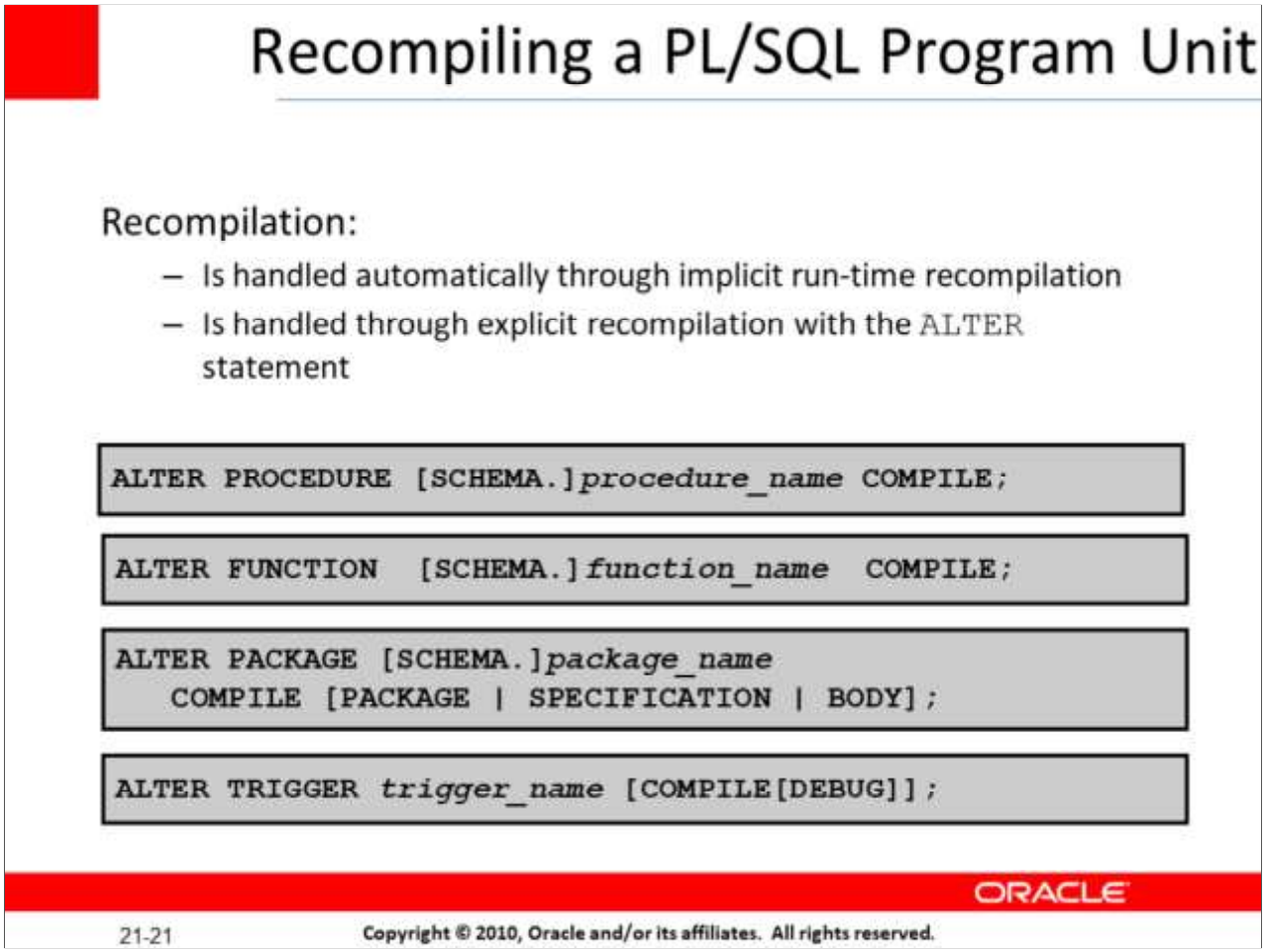
- Add columns to the table without invalidating dependent views or dependent PL/SQL objects
- Modify or delete columns not referenced by the view without invalidating dependent objects

Object Revalidation

The compiler cannot automatically revalidate an object that compiled with errors. The compiler recompiles the object, and if it recompiles without errors, it is revalidated; otherwise, it remains invalid.

The compiler checks whether the unauthorized object has access privileges to all of its referenced objects. If so, the compiler revalidates the unauthorized object without recompiling it. If not, the compiler issues appropriate error messages.

The SQL compiler recompiles the invalid object. If the object recompiles without errors, it is revalidated; otherwise, it remains invalid.



For an invalid PL/SQL program unit (procedure, function, or package), the PL/SQL compiler checks whether any referenced object changed in a way that affects the invalid object.

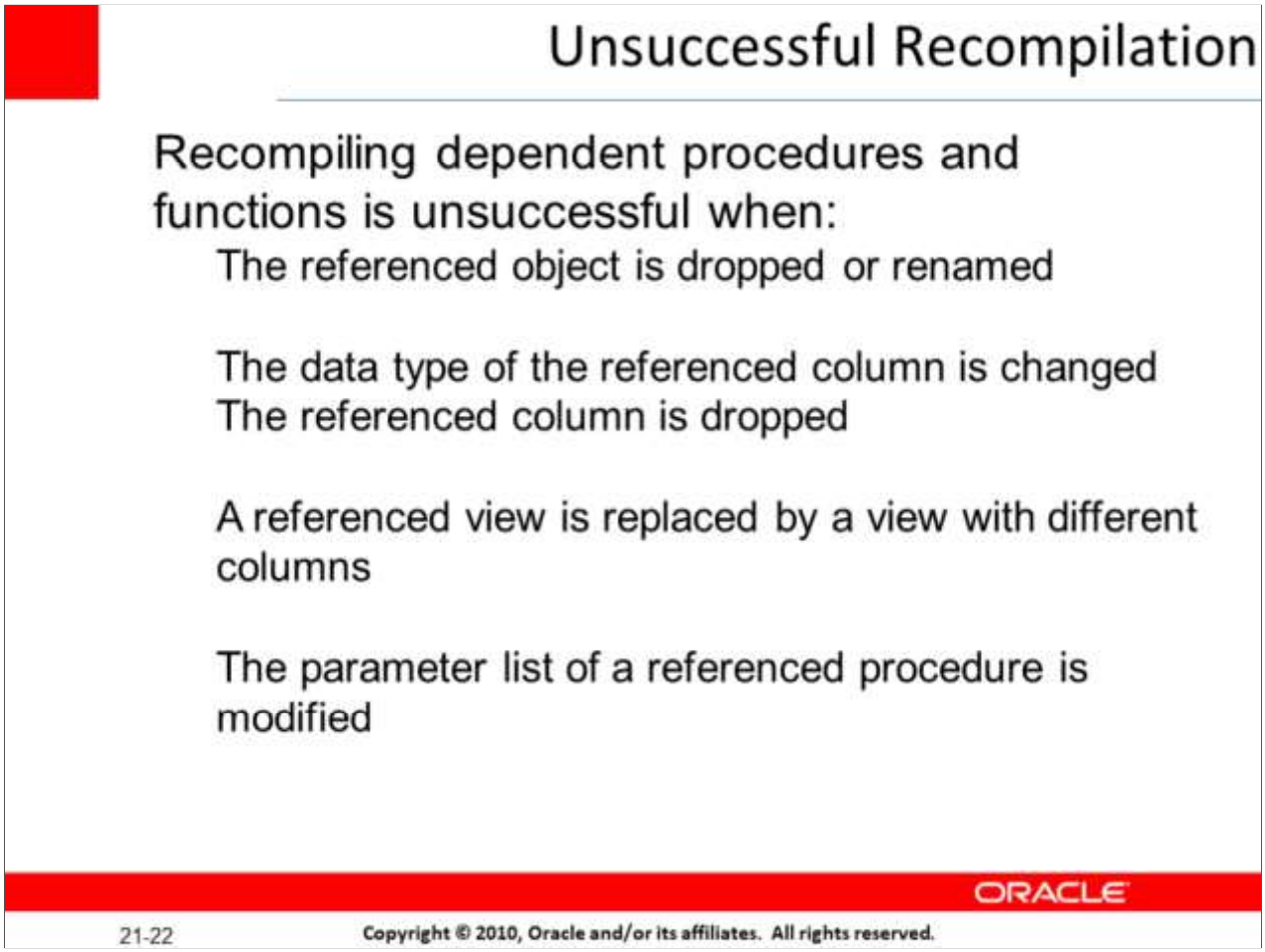
- If so, the compiler recompiles the invalid object. If the object recompiles without errors, it is revalidated; otherwise, it remains invalid. If not, the compiler revalidates the invalid object without recompiling it.
- If not, the compiler revalidates the invalid object without recompiling it. Fast revalidation is usually performed on objects that were invalidated due to cascading invalidation.

Recompiling PL/SQL Objects

If the recompilation is successful, the object becomes valid. If not, the Oracle server returns an error and the object remains invalid. When you recompile a PL/SQL object, the Oracle server first recompiles any invalid object on which it depends.

Procedure: Any local objects that depend on a procedure (such as procedures that call the recompiled procedure or package bodies that define the procedures that call the recompiled procedure) are also invalidated.

Packages: The COMPILEPACKAGE option recompiles both the package specification and the body, regardless of whether it is invalid. The COMPILESPECIFICATION option recompiles the package specification. Recompiling a package specification invalidates any local objects that depend on the specification, such as subprograms that use the package. Note that the body of a



package also depends on its specification. The COMPILEBODY option recompiles only the package body.

Triggers: Explicit recompilation eliminates the need for implicit run-time recompilation and prevents associated run-time compilation errors and performance overhead.

The DEBUG option instructs the PL/SQL compiler to generate and store the code for use by the PL/SQL debugger.

Unsuccessful Recompilation

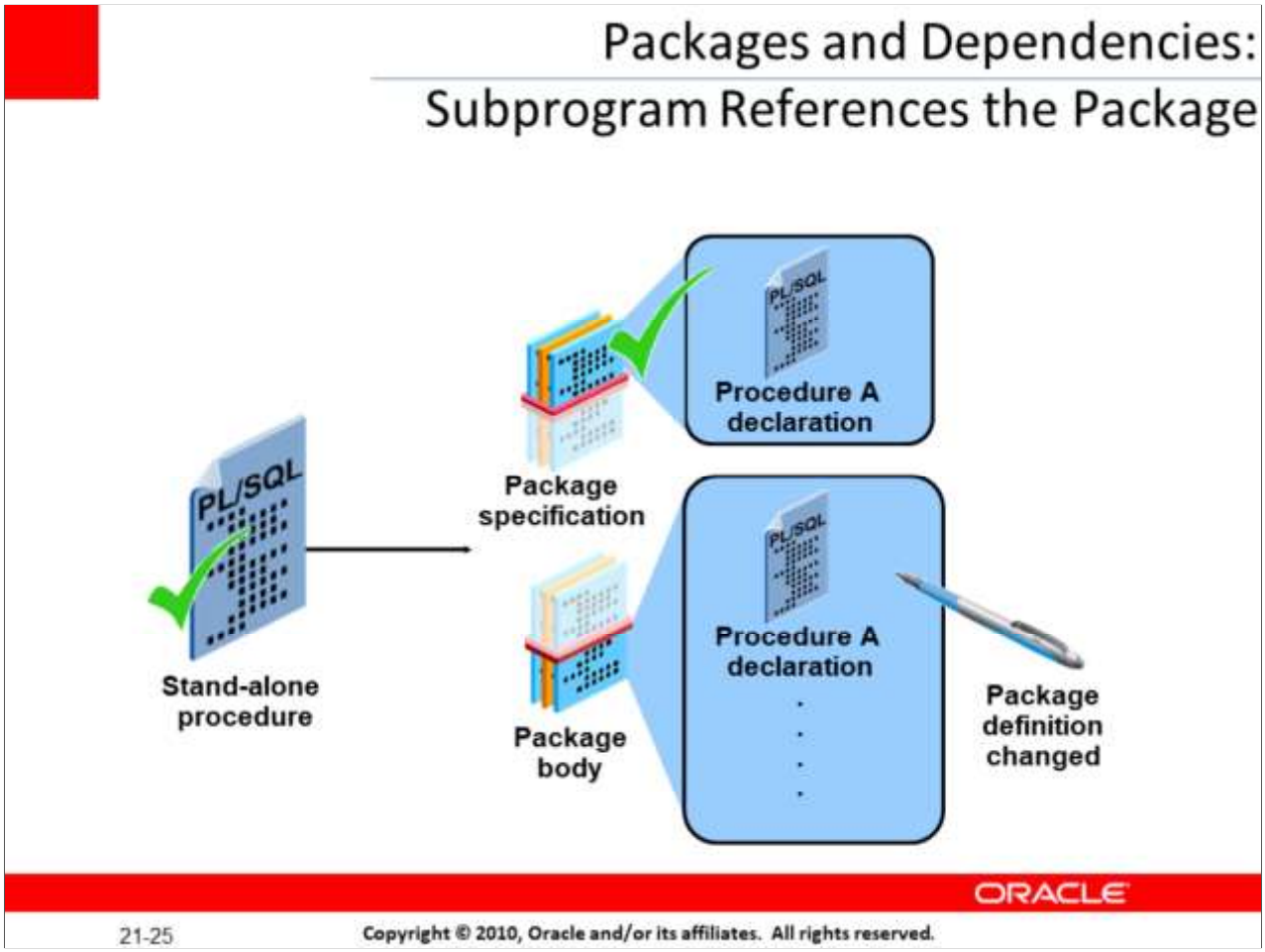
Sometimes a recompilation of dependent procedures is unsuccessful (for example, when a referenced table is dropped or renamed).

The success of any recompilation is based on the exact dependency. If a referenced view is recreated, any object that is dependent on the view needs to be recompiled. The success of the recompilation depends on the columns that the view now contains, as well as the columns that the dependent objects require for their execution. If the required columns are not part of the new view, then the object remains invalid.

Successful Recompilation

The recompilation of dependent objects is successful if:

- New columns are added to a referenced table
- All INSERT statements include a column list



- No new column is defined as NOTNULL

When a private table is referenced by a dependent procedure and the private table is dropped, the status of the dependent procedure becomes invalid. When the procedure is recompiled (either explicitly or implicitly) and a public table exists, the procedure can recompile successfully but is now dependent on the public table. The recompilation is successful only if the public table contains the columns that the procedure requires; otherwise, the status of the procedure remains invalid.

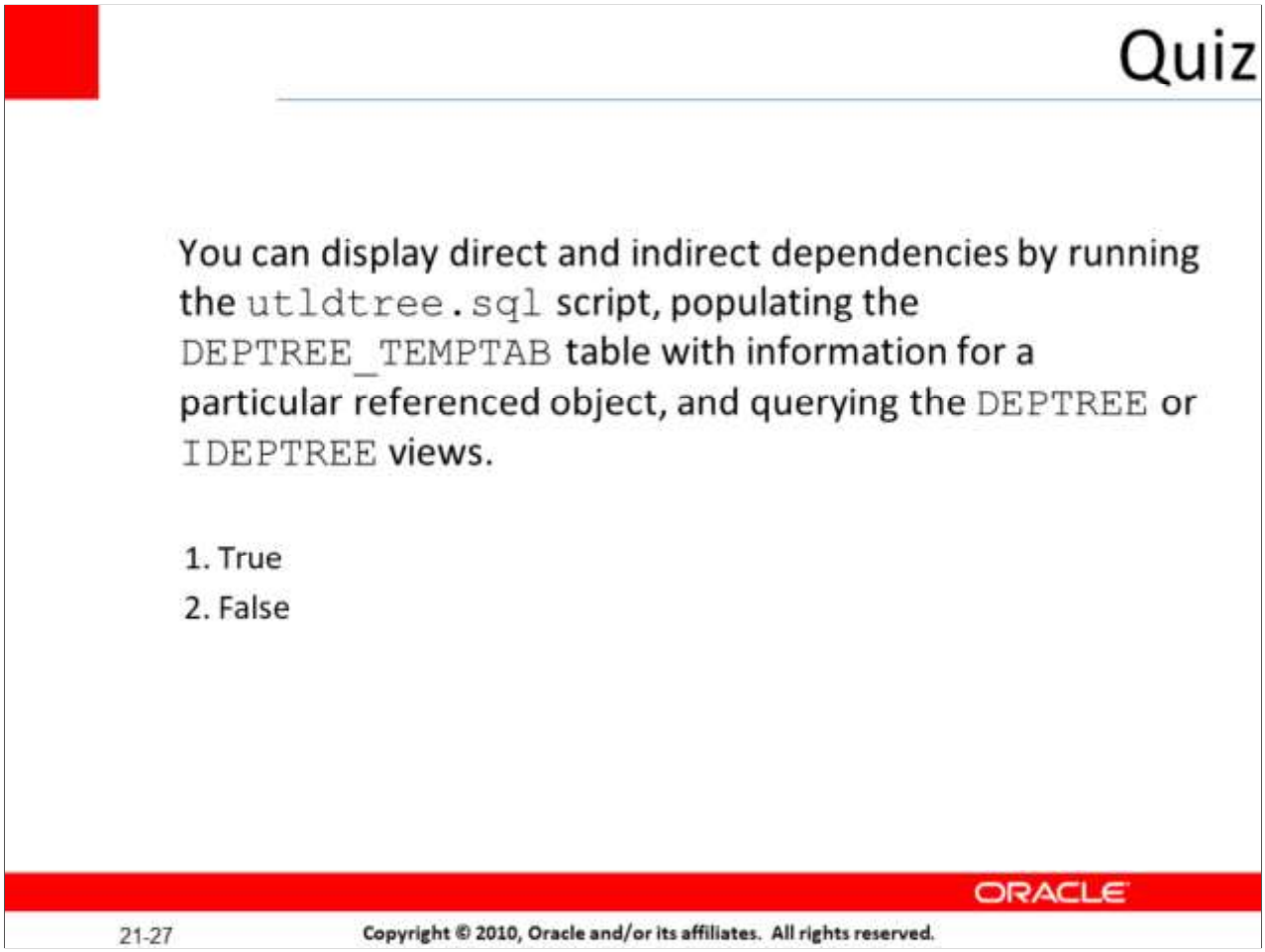
Recompilation of Procedures

You can minimize recompilation failure by following the guidelines that are shown in the slide.

Packages and Dependencies: Subprogram References the Package

You can simplify dependency management with packages when referencing a package procedure or function from a stand-alone procedure or function.

- If the package body changes and the package specification does not change, then the standalone procedure that references a package construct remains valid.
- If the package specification changes, then the outside procedure referencing a package construct is invalidated, as is the package body.



Packages and Dependencies: Package Subprogram References Procedure

If a stand-alone procedure that is referenced within the package changes, then the entire package body is invalidated, but the package specification remains valid. Therefore, it is recommended that you bring the procedure into the package.

Answer: 1 Displaying Direct and Indirect Dependencies

You can display direct and indirect dependencies as follows:

1. Run the utldtree.sql script, which creates the objects that enable you to display the direct and indirect dependencies.
2. Populate the DEPTREE_TEMPTAB table with information for a particular referenced object by executing the DEPTREE_FILL procedure.
3. Query the DEPTREE or IDEPTREE views.