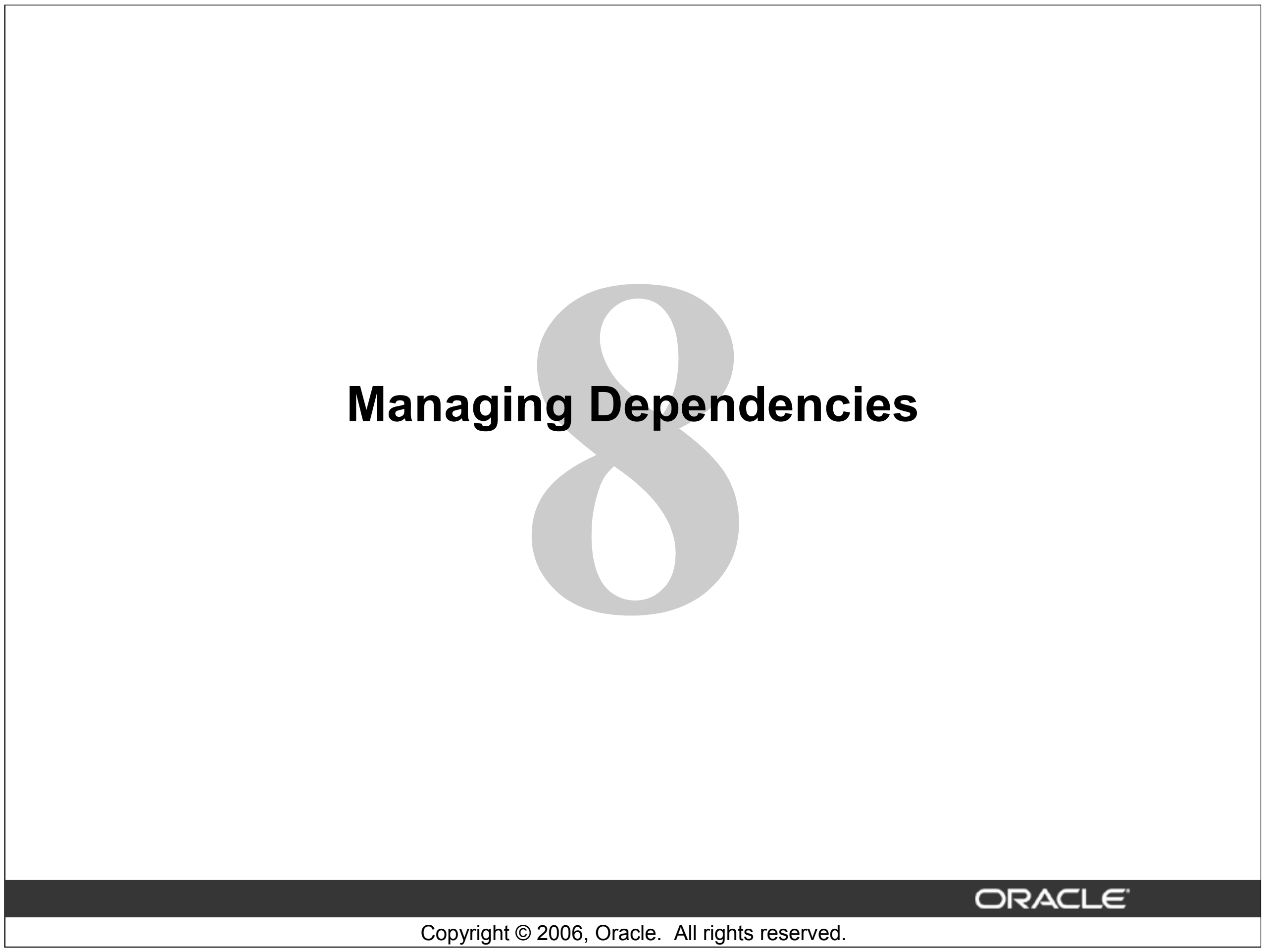


8

Managing Dependencies

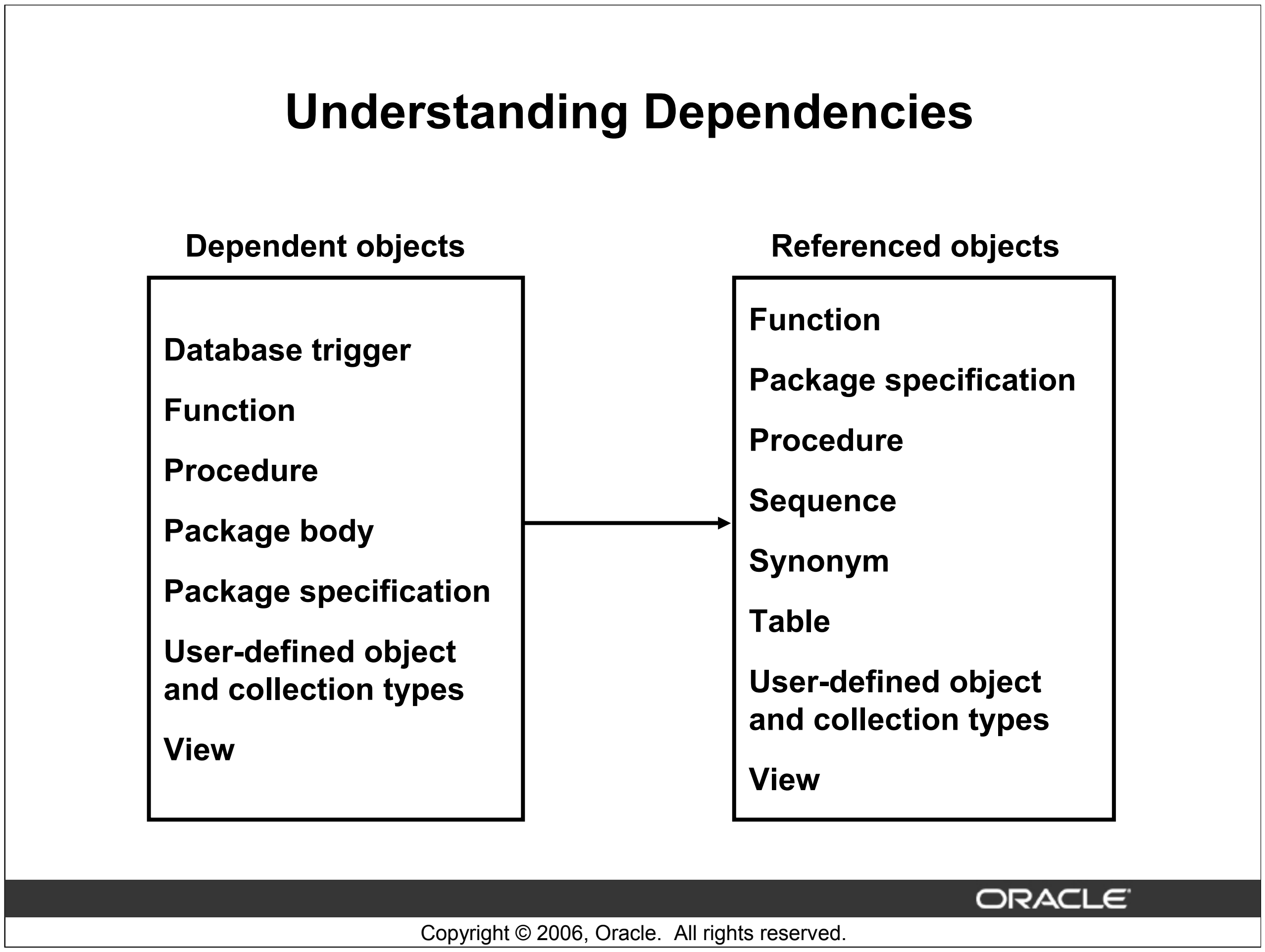
Objectives

After completing this lesson, you should be able to do the following:

- Track procedural dependencies
- Predict the effect of changing a database object on stored procedures and functions
- Manage procedural dependencies

Lesson Aim

This lesson introduces you to object dependencies and implicit and explicit recompilation of invalid objects.



Understanding Dependencies

Dependent objects	Referenced objects
Database trigger	Function
Function	Package specification
Procedure	Procedure
Package body	Sequence
Package specification	Synonym
User-defined object and collection types	Table
View	User-defined object and collection types
	View

Dependent and Referenced Objects

Some objects reference other objects as part of their definitions. For example, a stored procedure could contain a `SELECT` statement that selects columns from a table. For this reason, the stored procedure is called a dependent object, whereas the table is called a referenced object.

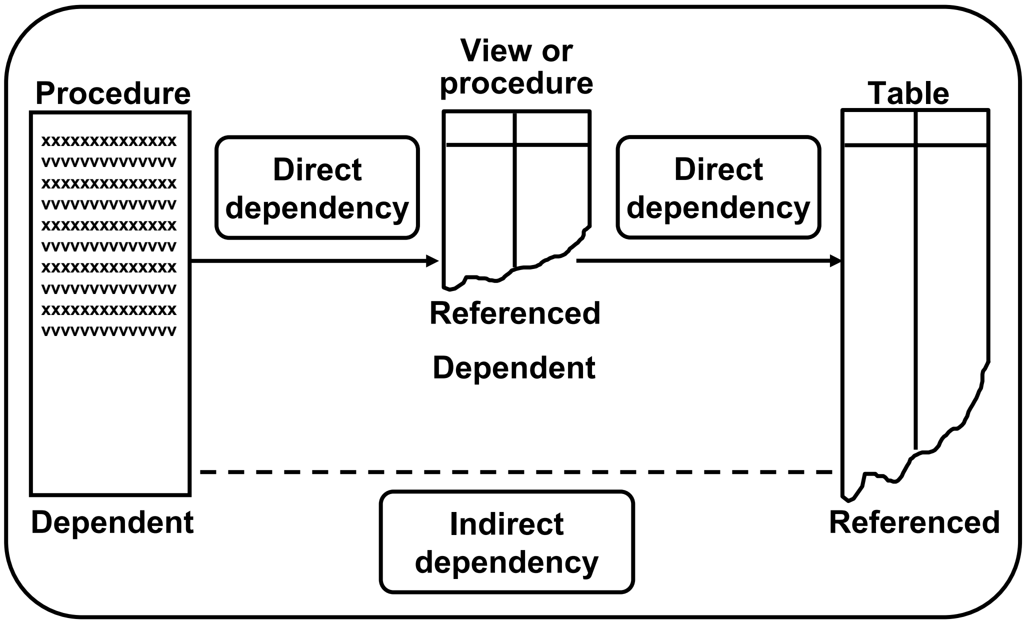
Dependency Issues

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.

Status	Significance
`VALID`	The schema object has been compiled and can be immediately used when referenced.
`INVALID`	The schema object must be compiled before it can be used.

Dependencies

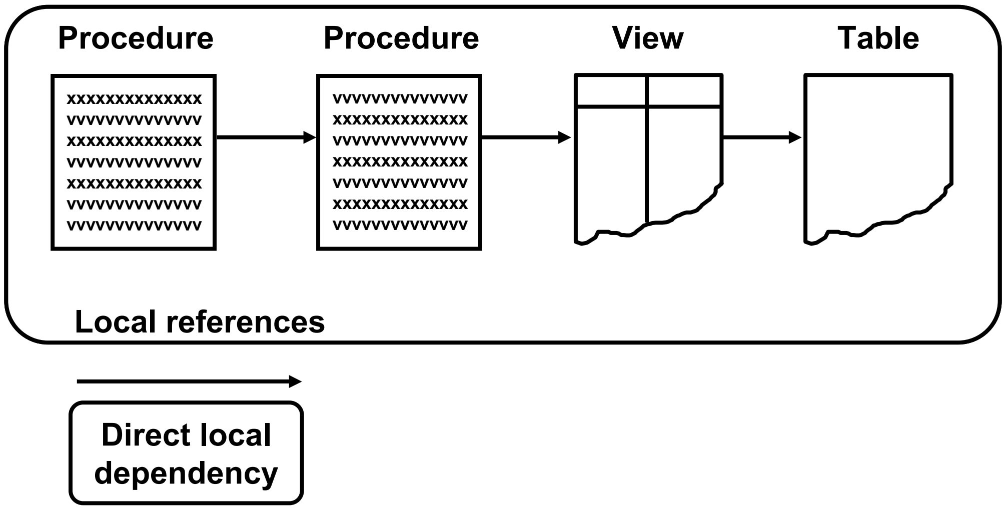


Dependent and Referenced Objects (continued)

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

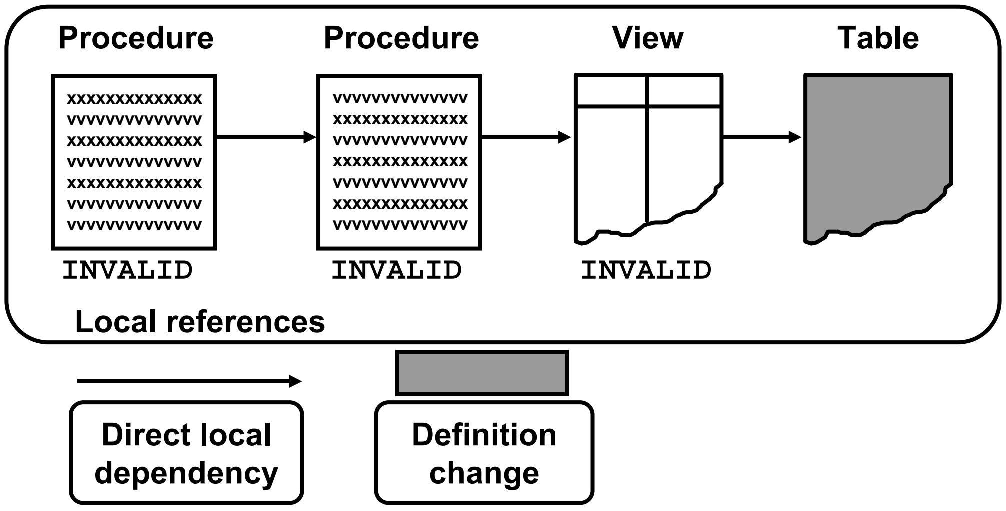
Local Dependencies



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 invalidated. The next time an invalidated object is called, the Oracle server automatically recompiles it.

Local Dependencies

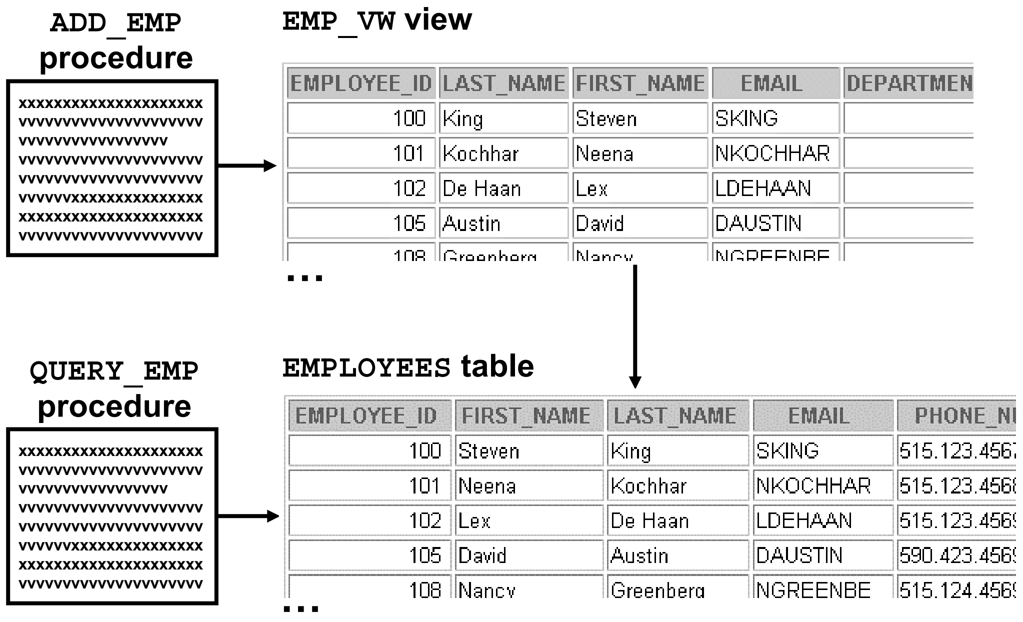


The Oracle server implicitly recompiles any `INVALID` object when the object is next called.

Managing Local Dependencies (continued)

Assume that the structure of the table on which a view is based is modified. When you describe the view by using the *i*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.

A Scenario of Local Dependencies



EMPLOYEE_ID	LAST_NAME	FIRST_NAME	EMAIL	DEPARTMEN
100	King	Steven	SKING	
101	Kochhar	Neena	NKOCHHAR	
102	De Haan	Lex	LDEHAAN	
105	Austin	David	DAUSTIN	
108	Greenberg	Nancy	NGREENBE	

…

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NU
100	Steven	King	SKING	515.123.456
101	Neena	Kochhar	NKOCHHAR	515.123.456
102	Lex	De Haan	LDEHAAN	515.123.456
105	David	Austin	DAUSTIN	590.423.456
108	Nancy	Greenberg	NGREENBE	515.124.456

…

ORACLE

Example

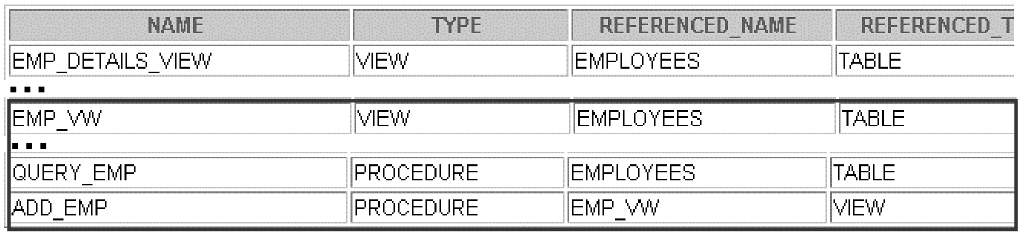
The QUERY_EMP procedure directly references the EMPLOYEES table. The ADD_EMP procedure updates the EMPLOYEES table indirectly by using the EMP_VW view.

In each of the following cases, is the ADD_EMP procedure invalidated and does it successfully recompile?

1. The internal logic of the QUERY_EMP procedure is modified.
2. A new column is added to the EMPLOYEES table.
3. The EMP_VW view is dropped.

Displaying Direct Dependencies by Using `USER_DEPENDENCIES`

```
SELECT name, type, referenced_name, referenced_type
FROM   user_dependencies
WHERE  referenced_name IN ('EMPLOYEES','EMP_VW' );
```



NAME	TYPE	REFERENCED_NAME	REFERENCED_T
EMP_DETAILS_VIEW	VIEW	EMPLOYEES	TABLE
...			
EMP_VW	VIEW	EMPLOYEES	TABLE
...			
QUERY_EMP	PROCEDURE	EMPLOYEES	TABLE
ADD_EMP	PROCEDURE	EMP_VW	VIEW

Displaying Direct Dependencies by Using `USER_DEPENDENCIES`

Determine which database objects to recompile manually by displaying direct dependencies from the `USER_DEPENDENCIES` data dictionary view.

Examine the `ALL_DEPENDENCIES` and `DBA_DEPENDENCIES` views, each of which contains the additional column `OWNER`, which references the owner of the object.

Column	**Column Description**
`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

Displaying Direct and Indirect Dependencies

1. **Run the `utldtree.sql` script that creates the objects that enable you to display the direct and indirect dependencies.**
2. **Execute the `DEPTREE_FILL` procedure.**

```
EXECUTE deptree_fill('TABLE','SCOTT','EMPLOYEES')
```

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

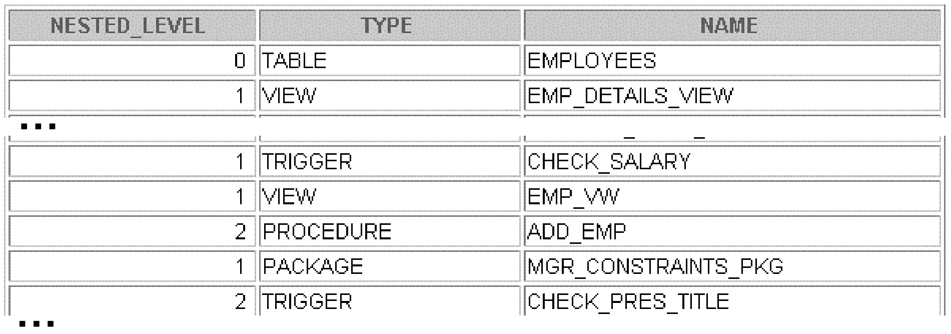
1. Make sure that the `utldtree.sql` script has been executed. This script is located in the `$ORACLE_HOME/rdbms/admin` folder. (This script is supplied in the `lab` folder of your class files.)
2. Populate the `DEPTREE_TEMPTAB` table with information for a particular referenced object by invoking the `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

The `DEPTREE` view:

```
SELECT   nested_level, type, name
FROM     deptree
ORDER BY seq#;
```



NESTED_LEVEL	TYPE	NAME
0	TABLE	EMPLOYEES
1	VIEW	EMP_DETAILS_VIEW
...		
1	TRIGGER	CHECK_SALARY
1	VIEW	EMP_VW
2	PROCEDURE	ADD_EMP
1	PACKAGE	MGR_CONSTRAINTS_PKG
2	TRIGGER	CHECK_PRES_TITLE
...		

Displaying Dependencies

Example

Display a tabular representation of all dependent objects by querying the DEPTREE view.

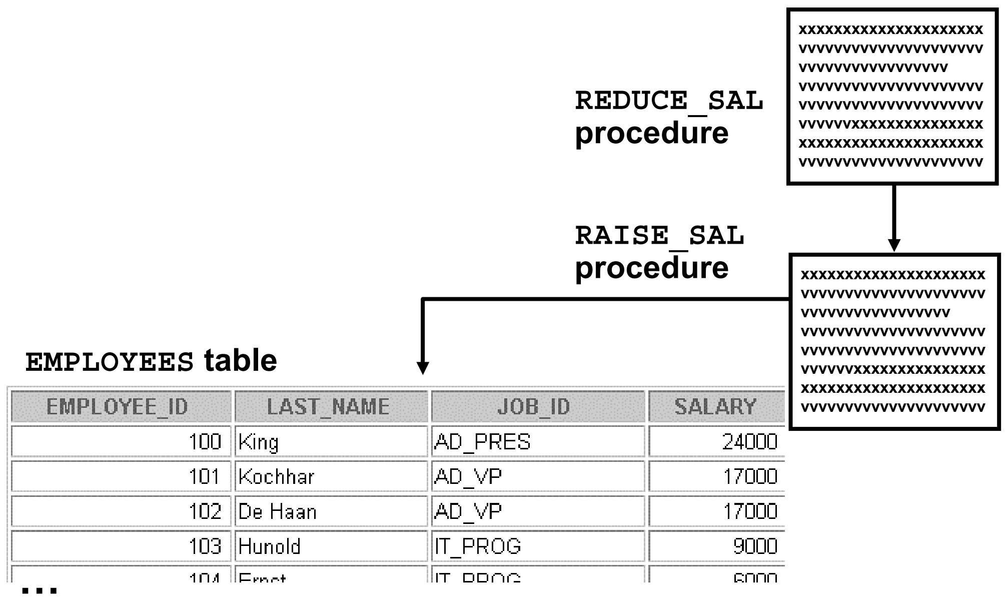
Display an indented representation of the same information by querying the IDEPTREE view, which consists of a single column named DEPENDENCIES.

For example,

```
SELECT *
FROM   ideptree;
```

provides a single column of indented output of the dependencies in a hierarchical structure.

Another Scenario of Local Dependencies



EMPLOYEE_ID	LAST_NAME	JOB_ID	SALARY
100	King	AD_PRES	24000
101	Kochhar	AD_VP	17000
102	De Haan	AD_VP	17000
103	Hunold	IT_PROG	9000
104	Ernst	IT_PROG	6000

...

Another Scenario of Local Dependencies

Example 1

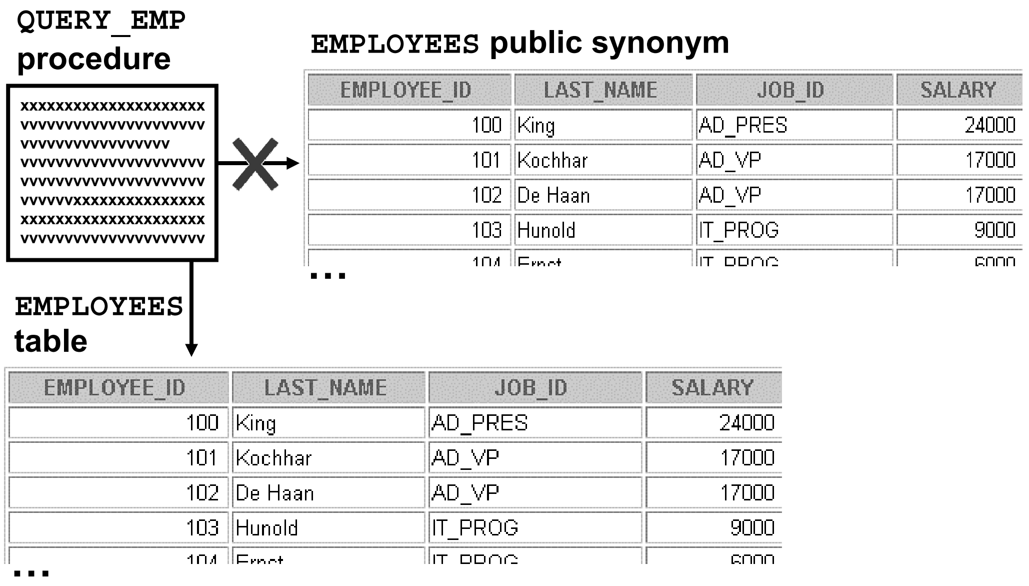
Predict the effect that a change in the definition of a procedure has on the recompilation of a dependent procedure.

Suppose 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.

In each of the following cases, does the REDUCE_SAL procedure successfully recompile?

1. The internal logic of the RAISE_SAL procedure is modified.
2. One of the formal parameters to the RAISE_SAL procedure is eliminated.

A Scenario of Local Naming Dependencies



EMPLOYEES public synonym

EMPLOYEE_ID	LAST_NAME	JOB_ID	SALARY
100	King	AD_PRES	24000
101	Kochhar	AD_VP	17000
102	De Haan	AD_VP	17000
103	Hunold	IT_PROG	9000
104	Ernst	IT_PROG	6000

...

EMPLOYEES table

EMPLOYEE_ID	LAST_NAME	JOB_ID	SALARY
100	King	AD_PRES	24000
101	Kochhar	AD_VP	17000
102	De Haan	AD_VP	17000
103	Hunold	IT_PROG	9000
104	Ernst	IT_PROG	6000

...

A Scenario of Local Naming Dependencies

Example 2

Be aware of the subtle case in which the creation of a table, view, or synonym may unexpectedly invalidate a dependent object because it interferes with the Oracle server hierarchy for resolving name references.

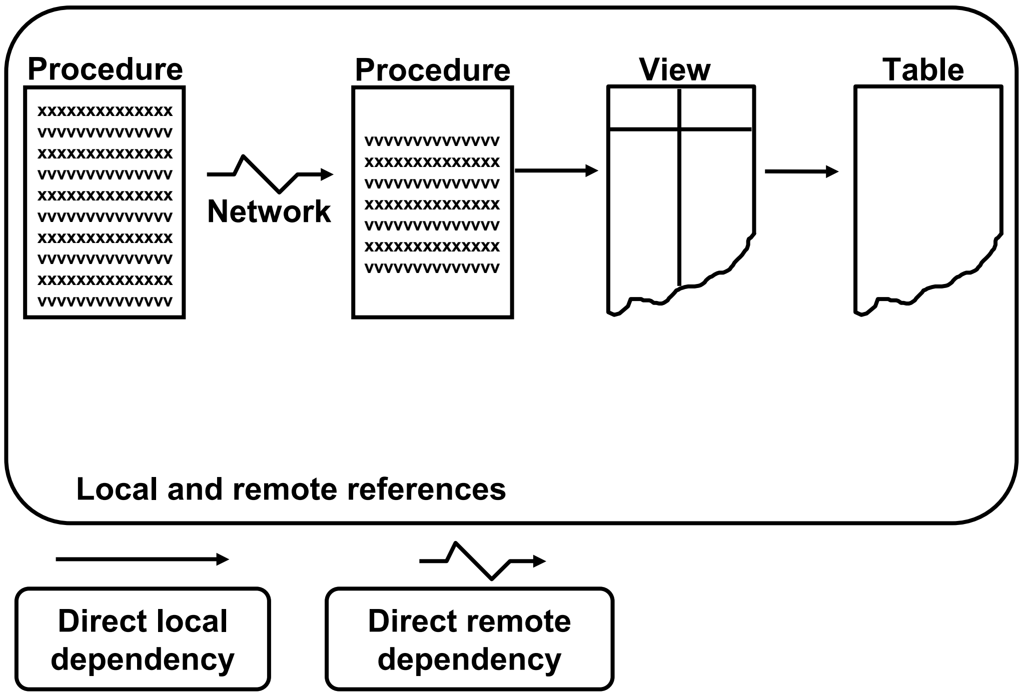
Predict the effect that the name of a new object has upon a dependent procedure.

Suppose that your QUERY_EMP procedure originally referenced a public synonym called EMPLOYEES. However, you have just created a new table called EMPLOYEES within your own schema. Does this change invalidate the procedure? Which of the two EMPLOYEES objects does QUERY_EMP reference when the procedure recompiles?

Now suppose that you drop your private EMPLOYEES table. Does this invalidate the procedure? What happens when the procedure recompiles?

You can track security dependencies in the USER_TAB_PRIVS data dictionary view.

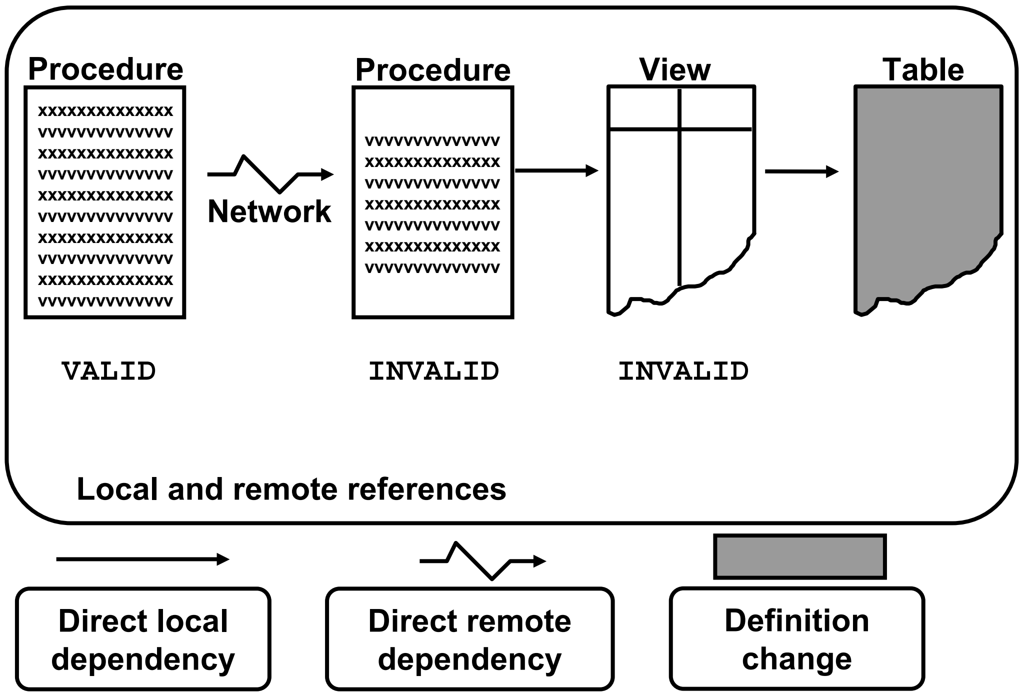
Understanding Remote Dependencies



Understanding Remote Dependencies

In the case of remote dependencies, the objects are on separate nodes. The Oracle server does not manage dependencies among remote schema objects other than local-procedure-to-remote-procedure dependencies (including functions, packages, and triggers). The local stored procedure and all its dependent objects are invalidated but do not automatically recompile when called for the first time.

Understanding Remote Dependencies



Understanding Remote Dependencies (continued)

Recompilation of Dependent Objects: Local and Remote

- Verify successful explicit recompilation of the dependent remote procedures and implicit recompilation of the dependent local procedures by checking the status of these procedures within the `USER_OBJECTS` view.
- If an automatic implicit recompilation of the dependent local procedures fails, the status remains invalid and the Oracle server issues a run-time error. Therefore, to avoid disrupting production, it is strongly recommended that you recompile local dependent objects manually, rather than relying on an automatic mechanism.

Concepts of Remote Dependencies

Remote dependencies are governed by the mode that is chosen by the user:

- **`TIMESTAMP` checking**
- **`SIGNATURE` checking**

Concepts of Remote Dependencies

`TIMESTAMP` Checking

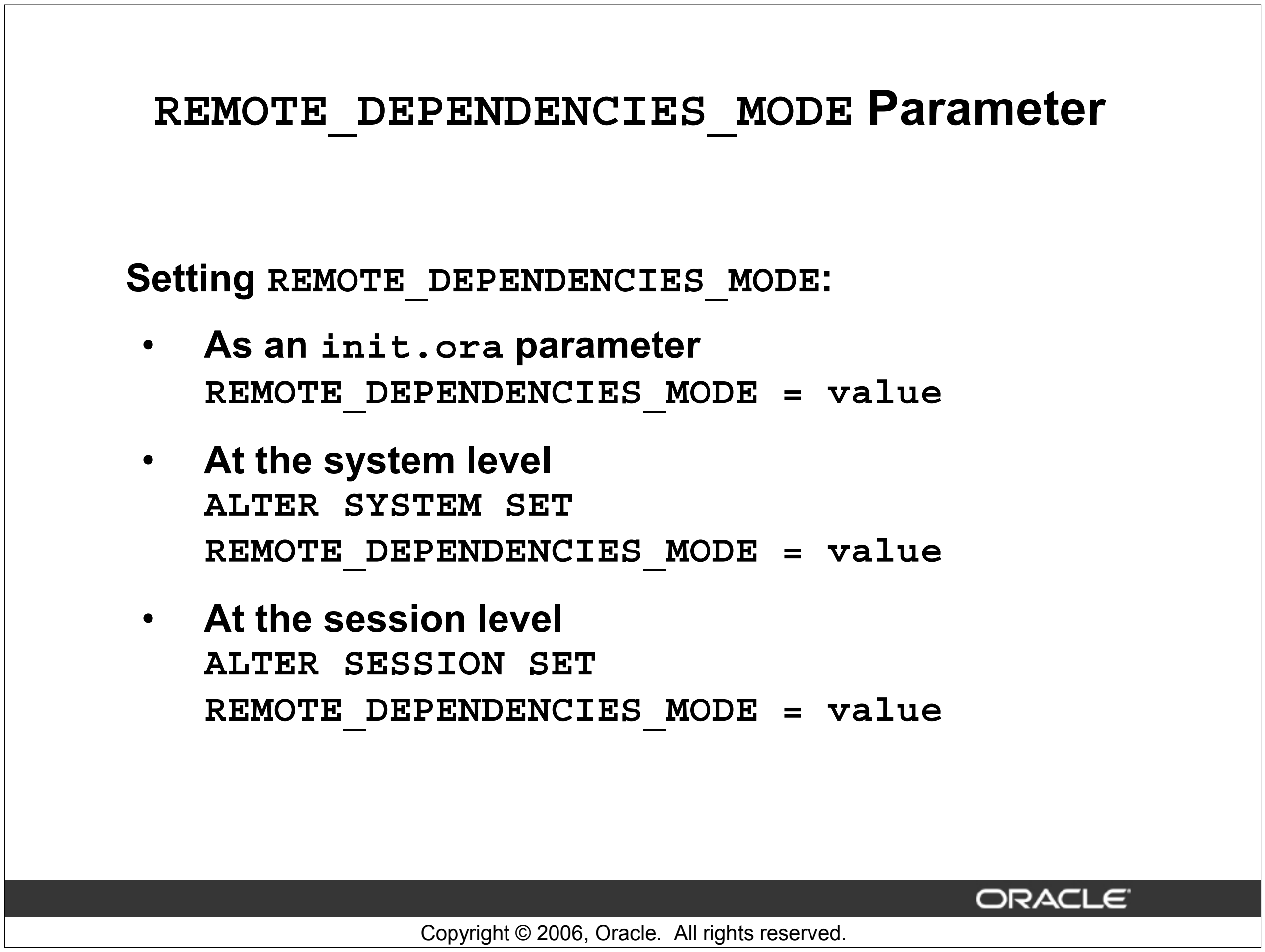
Each PL/SQL program unit carries a time stamp that is set when it is created or recompiled. Whenever you alter a PL/SQL program unit or a relevant schema object, all its dependent program units are marked as invalid and must be recompiled before they can execute. The actual time stamp comparison occurs when a statement in the body of a local procedure calls a remote procedure.

`SIGNATURE` Checking

For each PL/SQL program unit, both the time stamp and the signature are recorded. The signature of a PL/SQL construct contains information about the following:

- The name of the construct (procedure, function, or package)
- The base types of the parameters of the construct
- The modes of the parameters (`IN`, `OUT`, or `IN OUT`)
- The number of the parameters

The recorded time stamp in the calling program unit is compared with the current time stamp in the called remote program unit. If the time stamps match, the call proceeds. If they do not match, the remote procedure call (RPC) layer performs a simple comparison of the signature to determine whether the call is safe or not. If the signature has not been changed in an incompatible manner, execution continues; otherwise, an error is returned.



REMOTE_DEPENDENCIES_MODE Parameter

Setting REMOTE_DEPENDENCIES_MODE:

- As an init.ora parameter
  `REMOTE_DEPENDENCIES_MODE = value`
- At the system level
  ```
  ALTER SYSTEM SET
  REMOTE_DEPENDENCIES_MODE = value
  ```
- At the session level
  ```
  ALTER SESSION SET
  REMOTE_DEPENDENCIES_MODE = value
  ```

REMOTE_DEPENDENCIES_MODE Parameter

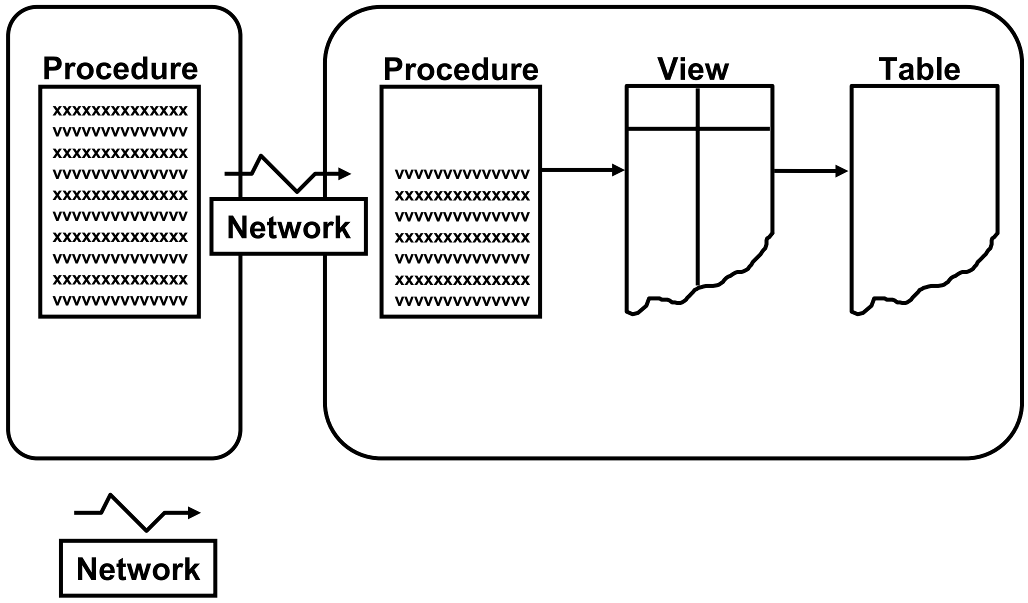
Setting the REMOTE_DEPENDENCIES_MODE

value	TIMESTAMP
	SIGNATURE

Specify the value of the REMOTE_DEPENDENCIES_MODE parameter using one of the three methods described in the slide.

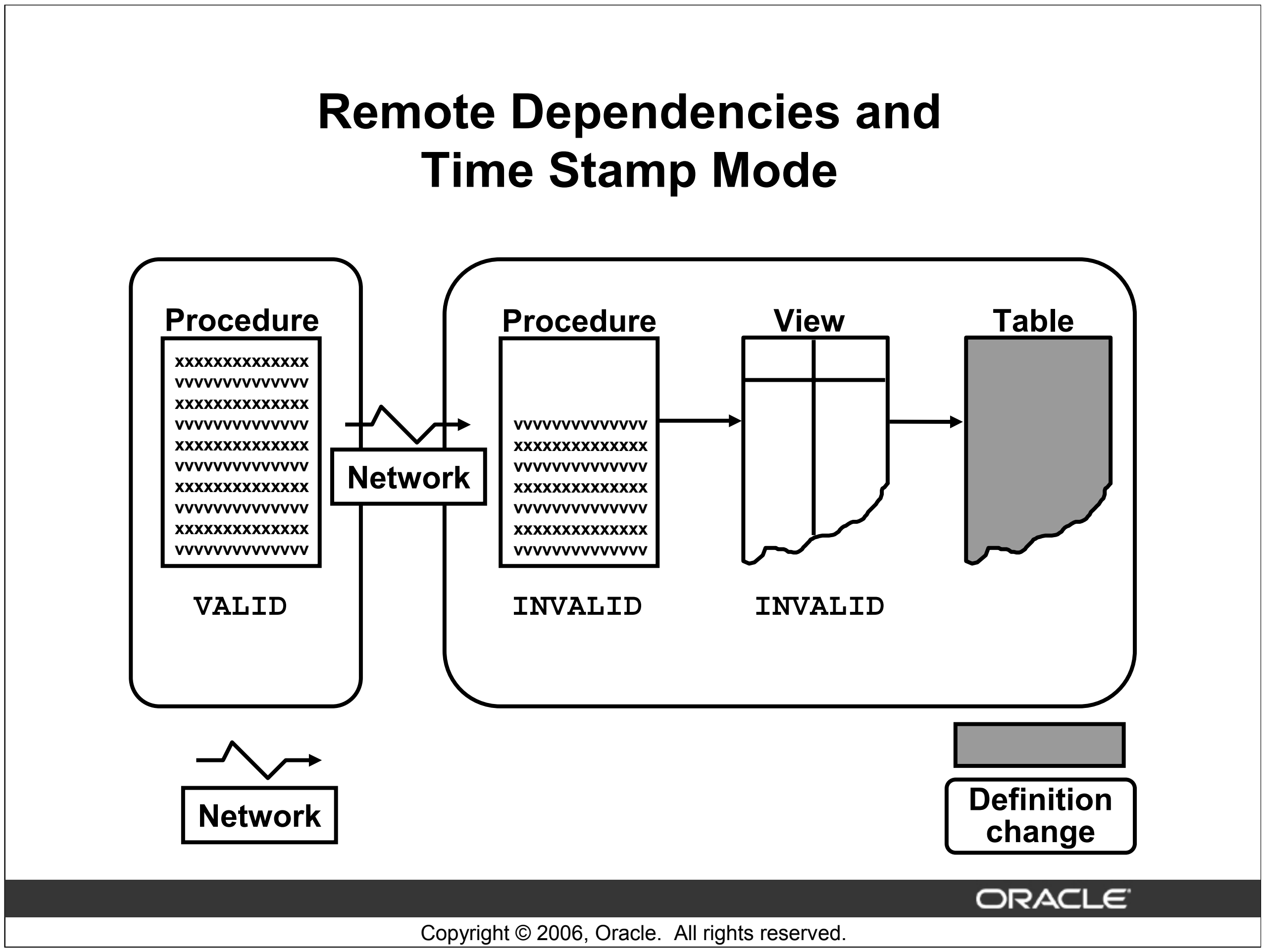
Note: The calling site determines the dependency model.

Remote Dependencies and Time Stamp Mode

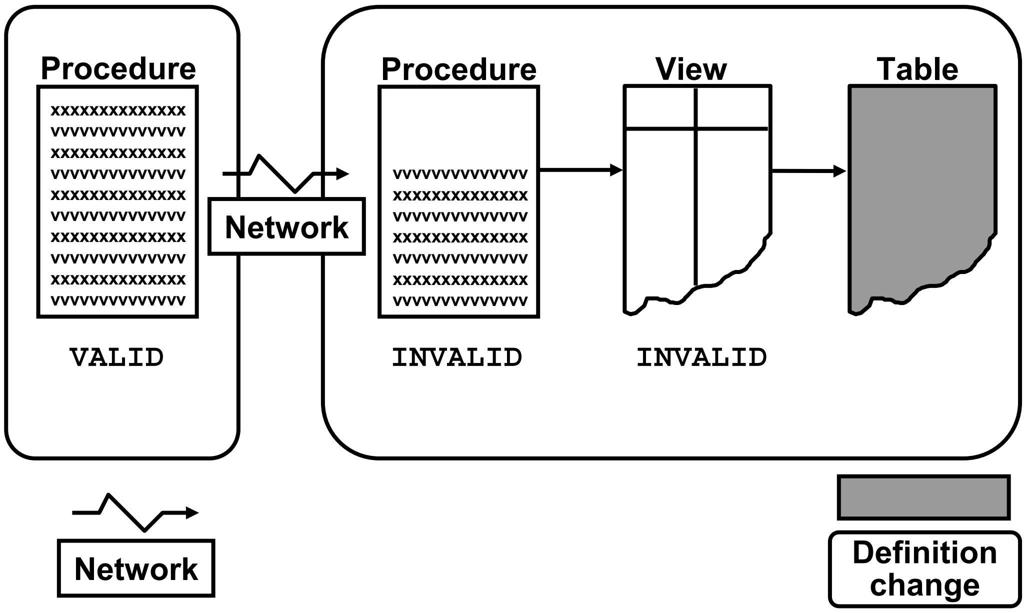


Remote Dependencies and Time Stamp Mode

If time stamps are used to handle dependencies among PL/SQL program units, then whenever you alter a program unit or a relevant schema object, all its dependent units are marked as invalid and must be recompiled before they can be run.



Remote Dependencies and Time Stamp Mode



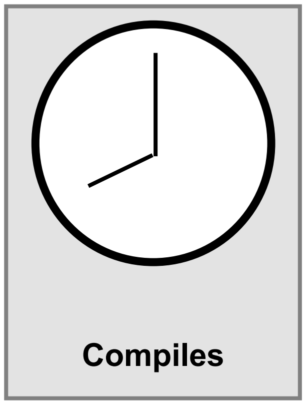
Remote Dependencies and Time Stamp Mode (continued)

In the example in the slide, the definition of the table changes. Therefore, all its dependent units are marked as invalid and must be recompiled before they can be run.

- When remote objects change, it is strongly recommended that you recompile local dependent objects manually in order to avoid disrupting production.
- The remote dependency mechanism is different from the automatic local dependency mechanism already discussed. The first time a recompiled remote subprogram is invoked by a local subprogram, you get an execution error and the local subprogram is invalidated; the second time it is invoked, implicit automatic recompilation takes place.

Remote Procedure B Compiles at 8:00 a.m.

Remote procedure B



Valid

Local Procedures Referencing Remote Procedures

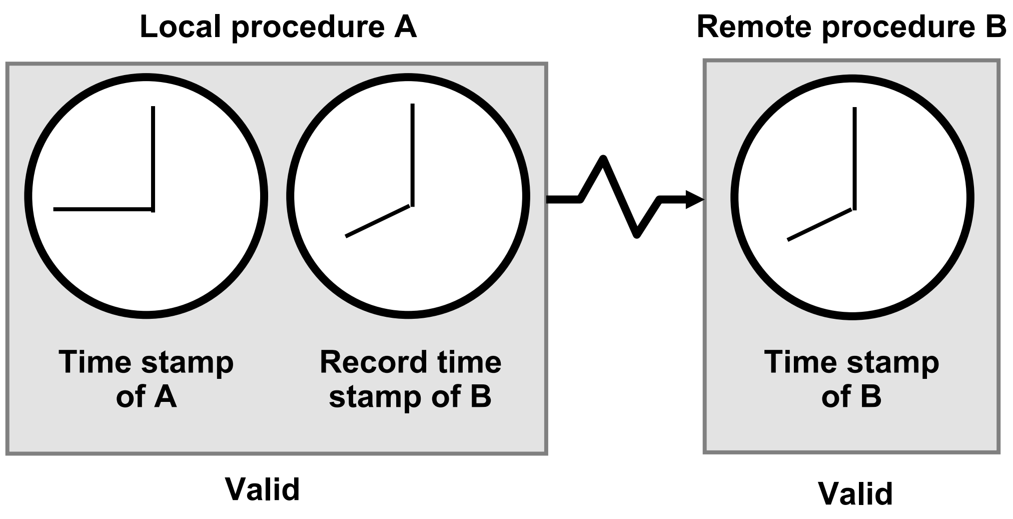
A local procedure that references a remote procedure is invalidated by the Oracle server if the remote procedure is recompiled after the local procedure is compiled.

Automatic Remote Dependency Mechanism

When a procedure compiles, the Oracle server records the time stamp of that compilation within the `P` code of the procedure.

In the slide, when the remote procedure B is successfully compiled at 8:00 a.m., this time is recorded as its time stamp.

Local Procedure A Compiles at 9:00 a.m.



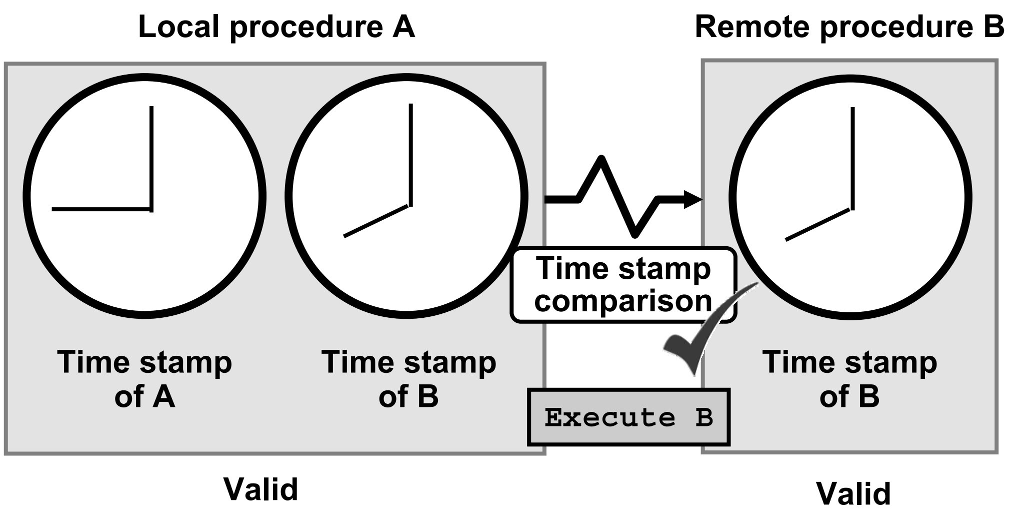
Local Procedures Referencing Remote Procedures (continued)

Automatic Remote Dependency Mechanism (continued)

When a local procedure referencing a remote procedure compiles, the Oracle server also records the time stamp of the remote procedure in the `P` code of the local procedure.

In the slide, local procedure A (which is dependent on remote procedure B) is compiled at 9:00 a.m. The time stamps of both procedure A and remote procedure B are recorded in the `P` code of procedure A.

Execute Procedure A



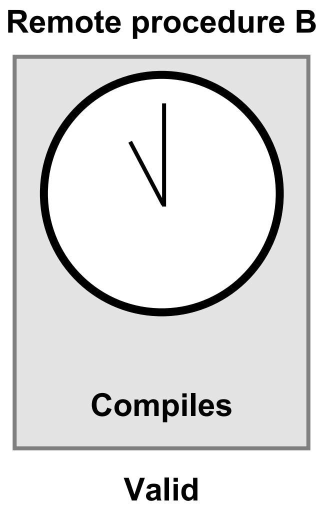
Automatic Remote Dependency

When the local procedure is invoked at run time, the Oracle server compares the two time stamps of the referenced remote procedure.

If the time stamps are equal (indicating that the remote procedure has not recompiled), then the Oracle server executes the local procedure.

In the example in the slide, the time stamp recorded with the `P` code of remote procedure B is the same as that recorded with local procedure A. Therefore, local procedure A is valid.

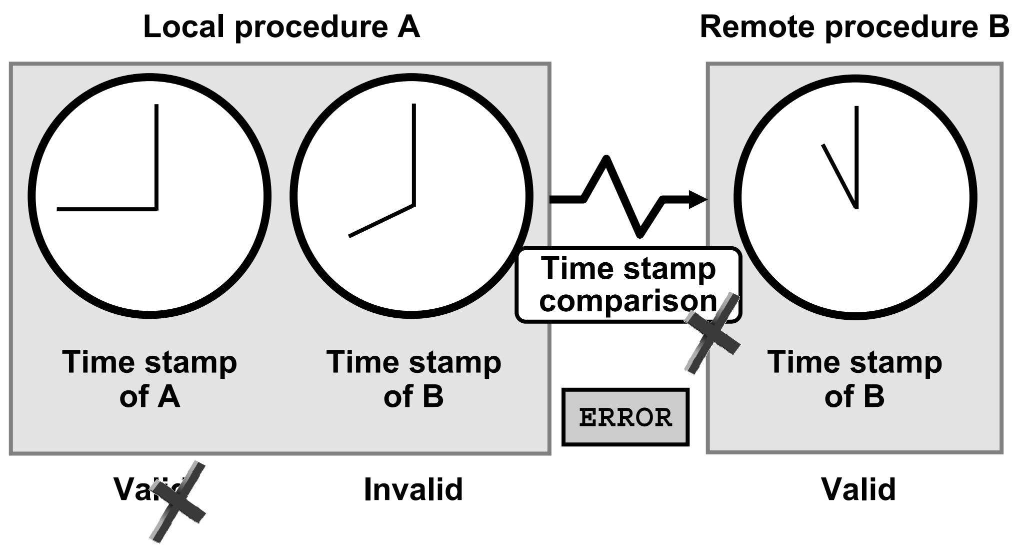
Remote Procedure B Recompiled at 11:00 a.m.



Local Procedures Referencing Remote Procedures

Assume that remote procedure B is successfully recompiled at 11:00 a.m. The new time stamp is recorded along with its `P` code.

Execute Procedure A



Automatic Remote Dependency

If the time stamps are not equal (indicating that the remote procedure has recompiled), then the Oracle server invalidates the local procedure and returns a run-time error. If the local procedure (which is now tagged as invalid) is invoked a second time, then the Oracle server recompiles it before executing, in accordance with the automatic local dependency mechanism.

Note: If a local procedure returns a run-time error the first time it is invoked (indicating that the remote procedure's time stamp has changed), then you should develop a strategy to reinvoke the local procedure.

In the example in the slide, the remote procedure is recompiled at 11:00 a.m. and this time is recorded as its time stamp in the `P` code. The `P` code of local procedure A still has 8:00 a.m. as the time stamp for remote procedure B. Because the time stamp recorded with the `P` code of local procedure A is different from that recorded with the remote procedure B, the local procedure is marked invalid. When the local procedure is invoked for the second time, it can be successfully compiled and marked valid.

A disadvantage of time stamp mode is that it is unnecessarily restrictive. Recompilation of dependent objects across the network is often performed when not strictly necessary, leading to performance degradation.

Signature Mode

- The signature of a procedure is:
  - The name of the procedure
  - The data types of the parameters
  - The modes of the parameters
- The signature of the remote procedure is saved in the local procedure.
- When executing a dependent procedure, the signature of the referenced remote procedure is compared.

Signatures

To alleviate some of the problems with the time stamp–only dependency model, you can use the signature model. This allows the remote procedure to be recompiled without affecting the local procedures. This is important if the database is distributed.

The signature of a subprogram contains the following information:

- The name of the subprogram
- The data types of the parameters
- The modes of the parameters
- The number of parameters
- The data type of the return value for a function

If a remote program is changed and recompiled but the signature does not change, then the local procedure can execute the remote procedure. With the time stamp method, an error would have been raised because the time stamps would not have matched.

Recompiling a PL/SQL Program Unit

Recompilation:

- **Is handled automatically through implicit run-time recompilation**
- **Is handled through explicit recompilation with the `ALTER` statement**

```
ALTER PROCEDURE [SCHEMA.]procedure_name COMPILE;
```

```
ALTER FUNCTION  [SCHEMA.]function_name  COMPILE;
```

```
ALTER PACKAGE [SCHEMA.]package_name
   COMPILE [PACKAGE | SPECIFICATION | BODY];
```

```
ALTER TRIGGER trigger_name [COMPILE[DEBUG]];
```

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 `COMPILE PACKAGE` option recompiles both the package specification and the body, regardless of whether it is invalid. The `COMPILE SPECIFICATION` 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 `COMPILE BODY` 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

Recompiling dependent procedures and functions is unsuccessful when:

- **The referenced object is dropped or renamed**
- **The data type of the referenced column is changed**
- **The referenced column is dropped**
- **A referenced view is replaced by a view with different columns**
- **The parameter list of a referenced procedure is modified**

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 re-created, 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

Recompiling dependent procedures and functions is successful if:

- **The referenced table has new columns**
- **The data type of referenced columns has not changed**
- **A private table is dropped, but a public table that has the same name and structure exists**
- **The PL/SQL body of a referenced procedure has been modified and recompiled successfully**

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 `NOT NULL`

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

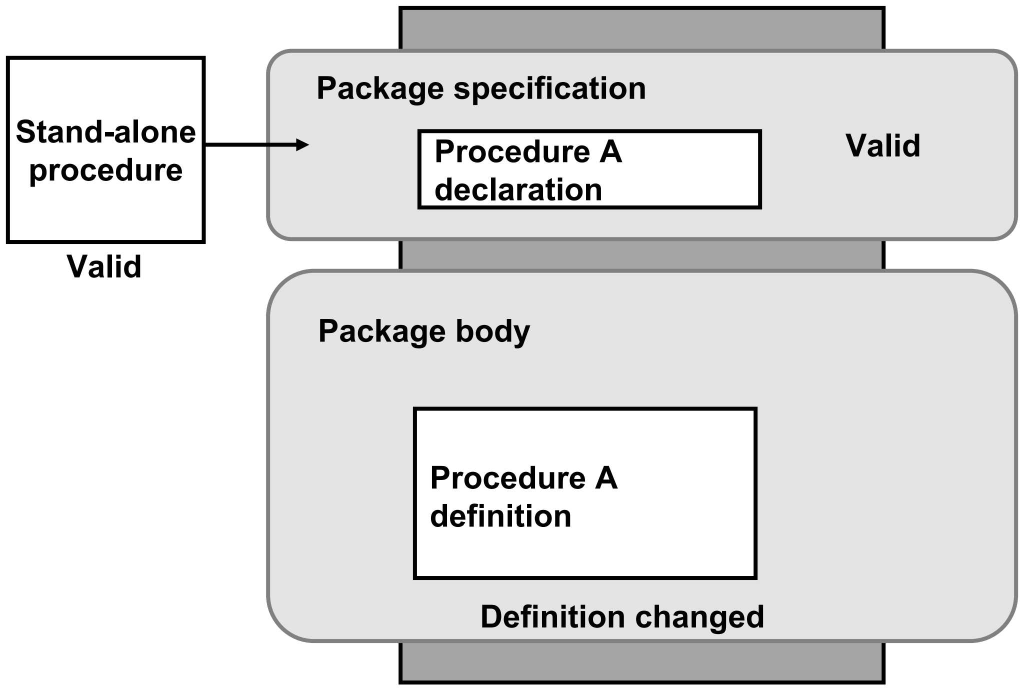
Minimize dependency failures by:

- Declaring records with the `%ROWTYPE` attribute
- Declaring variables with the `%TYPE` attribute
- Querying with the `SELECT *` notation
- Including a column list with `INSERT` statements

Recompilation of Procedures

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

Packages and Dependencies

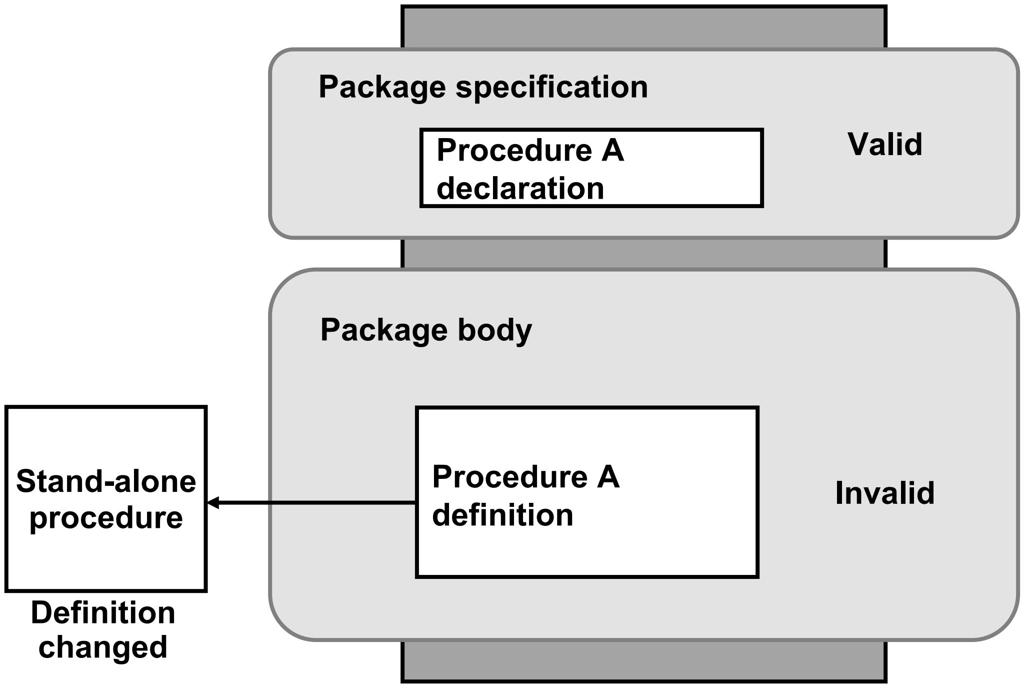


Managing Dependencies

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 stand-alone 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



Managing Dependencies (continued)

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.

Summary

In this lesson, you should have learned how to:

- Keep track of dependent procedures
- Recompile procedures manually as soon as possible after the definition of a database object changes

ORACLE

Summary

Avoid disrupting production by keeping track of dependent procedures and recompiling them manually as soon as possible after the definition of a database object changes.

Situation	**Automatic Recompilation**
Procedure depends on a local object.	Yes, at first reexecution
Procedure depends on a remote procedure.	Yes, but at second reexecution. Use manual recompilation for first reexecution, or reinvoke it a second time.
Procedure depends on a remote object other than a procedure.	No

Practice 8: Overview

This practice covers the following topics:

- **Using `DEPTREE_FILL` and `IDEPTREE` to view dependencies**
- **Recompiling procedures, functions, and packages**

Practice 8: Overview

In this practice, you use the `DEPTREE_FILL` procedure and the `IDEPTREE` view to investigate dependencies in your schema. In addition, you recompile invalid procedures, functions, packages, and views.

Practice 8

1. Answer the following questions:
   a. Can a table or a synonym be invalidated?
   b. Consider the following dependency example:
      The stand-alone procedure `MY_PROC` depends on the `MY_PROC_PACK` package procedure. The `MY_PROC_PACK` procedure's definition is changed by recompiling the package body. The `MY_PROC_PACK` procedure's declaration is not altered in the package specification.

      In this scenario, is the stand-alone procedure `MY_PROC` invalidated?
2. Create a tree structure showing all dependencies involving your `add_employee` procedure and your `valid_deptid` function.
   Note: `add_employee` and `valid_deptid` were created in the lesson titled "Creating Stored Functions." You can run the solution scripts for Practice 2 if you need to create the procedure and function.
   a. Load and execute the `utldtree.sql` script, which is located in the `E:\lab\PLPU\labs` folder.
   b. Execute the `deptree_fill` procedure for the `add_employee` procedure.
   c. Query the `IDEPTREE` view to see your results.
   d. Execute the `deptree_fill` procedure for the `valid_deptid` function.
   e. Query the `IDEPTREE` view to see your results.

If you have time, complete the following exercise:

3. Dynamically validate invalid objects.
   a. Make a copy of your `EMPLOYEES` table, called `EMPS`.
   b. Alter your `EMPLOYEES` table and add the column `TOTSAL` with data type `NUMBER(9,2).`
   c. Create and save a query to display the name, type, and status of all invalid objects.
   d. In the `compile_pkg` (created in Practice 6 in the lesson titled "Dynamic SQL and Metadata"), add a procedure called `recompile` that recompiles all invalid procedures, functions, and packages in your schema. Use Native Dynamic SQL to alter the invalid object type and compile it.
   e. Execute the `compile_pkg.recompile` procedure.
   f. Run the script file that you created in step 3c to check the status column value. Do you still have objects with an `INVALID` status?