Understanding Dependencies
What Will I Learn?

In this lesson, you will learn to:

- Describe the implications of procedural dependencies
- Contrast dependent objects and referenced objects
- View dependency information in the dictionary views
- Use the UTLDTREE script to create the objects required to display dependencies
- Use the IDEPTREE and DEPTREE views to display dependencies
- Describe when automatic recompilation occurs
- List how to minimize dependency failures
Why Learn It?

A PL/SQL subprogram can execute correctly only if the objects it references exist and are valid. These objects can be tables, views, other PL/SQL subprograms, and other kinds of database object.

So what happens if a referenced object is altered or dropped?

This lesson introduces you to object dependencies and implicit and explicit recompilation of invalid objects.
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.
Tell Me / Show Me

Dependencies

- Direct dependency
- Indirect dependency
- View or procedure
- Referenced
- Dependent
- Table
Tell Me / Show Me

Dependencies Summarized

<table>
<thead>
<tr>
<th>Dependent objects</th>
<th>Referenced objects</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table</td>
<td>Function</td>
</tr>
<tr>
<td>View</td>
<td>Package specification</td>
</tr>
<tr>
<td>Database trigger</td>
<td>Procedure</td>
</tr>
<tr>
<td>Procedure</td>
<td>Sequence</td>
</tr>
<tr>
<td>Function</td>
<td>Synonym</td>
</tr>
<tr>
<td>Package body</td>
<td>Table</td>
</tr>
<tr>
<td>Package specification</td>
<td>View</td>
</tr>
<tr>
<td>User-defined object and collection types</td>
<td>User-defined object and collection types</td>
</tr>
</tbody>
</table>
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 tries to recompile it.
The Oracle server implicitly attempts to recompile any INVALID object when the object is next called.
Tell Me / Show Me

A Scenario of Local Dependencies

**ADD_EMP**
- procedure

**QUERY_EMP**
- procedure

**EMP_VW**
- view

<table>
<thead>
<tr>
<th>EMPLOYEE_ID</th>
<th>LAST_NAME</th>
<th>FIRST_NAME</th>
<th>EMAIL</th>
<th>DEPARTMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>King</td>
<td>Steven</td>
<td>SKING</td>
<td>90</td>
</tr>
<tr>
<td>101</td>
<td>Kochhar</td>
<td>Neena</td>
<td>NKOCHHAR</td>
<td>90</td>
</tr>
<tr>
<td>102</td>
<td>De Haan</td>
<td>Lex</td>
<td>LDEHAAN</td>
<td>90</td>
</tr>
<tr>
<td>103</td>
<td>Hunold</td>
<td>Alexander</td>
<td>AHUNOLD</td>
<td>60</td>
</tr>
<tr>
<td>104</td>
<td>Ernst</td>
<td>Bruce</td>
<td>BERNST</td>
<td>60</td>
</tr>
</tbody>
</table>

**EMPLOYEES**
- table

<table>
<thead>
<tr>
<th>EMPLOYEE_ID</th>
<th>LAST_NAME</th>
<th>FIRST_NAME</th>
<th>EMAIL</th>
<th>PHONE_III</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>King</td>
<td>Steven</td>
<td>SKING</td>
<td>515.123.456</td>
</tr>
<tr>
<td>101</td>
<td>Kochhar</td>
<td>Neena</td>
<td>NKOCHHAR</td>
<td>515.123.456</td>
</tr>
<tr>
<td>102</td>
<td>De Haan</td>
<td>Lex</td>
<td>LDEHAAN</td>
<td>515.123.456</td>
</tr>
<tr>
<td>103</td>
<td>Hunold</td>
<td>Alexander</td>
<td>AHUNOLD</td>
<td>590.423.456</td>
</tr>
<tr>
<td>104</td>
<td>Ernst</td>
<td>Bruce</td>
<td>BERNST</td>
<td>590.423.456</td>
</tr>
</tbody>
</table>
Displaying Direct Dependencies by Using `USER_DEPENDENCIES`

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

<table>
<thead>
<tr>
<th>NAME</th>
<th>TYPE</th>
<th>REFERENCED_NAME</th>
<th>REFERENCED_TYPE</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD_EMP</td>
<td>PROCEDURE</td>
<td>EMP_VW</td>
<td>VIEW</td>
</tr>
<tr>
<td>EMP_VW</td>
<td>VIEW</td>
<td>EMPLOYEES</td>
<td>TABLE</td>
</tr>
<tr>
<td>QUERY_EMP</td>
<td>PROCEDURE</td>
<td>EMPLOYEES</td>
<td>TABLE</td>
</tr>
</tbody>
</table>

3 rows returned in 0.02 seconds

Direct Dependencies can also be viewed in Application Express: SQL Workshop -> Object Browser -> choose an object, then click the Dependencies tab.
Tell Me / Show Me

Displaying Direct and Indirect Dependencies

Run the script `utldtree.sql` that creates the objects that enable you to display the direct and indirect dependencies. This script creates four objects:

- A table `deptree_temptab` to hold dependency data
- A procedure `deptree_fill` to populate the table
- Two views `deptree` and `ideptree` to select and format dependency data from the populated table.

For each object whose dependencies you want to see:

1. Execute the `DEPTREE_FILL` procedure.

```
BEGIN deptree_fill('TABLE','SCOTT','EMPLOYEES'); END;
```
Tell Me / Show Me

Displaying Direct and Indirect Dependencies (continued)

2. Display the dependency data using the `DEPTREE` view

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

<table>
<thead>
<tr>
<th>NESTED_LEVEL</th>
<th>TYPE</th>
<th>NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>TABLE</td>
<td>EMPLOYEES</td>
</tr>
<tr>
<td>1</td>
<td>VIEW</td>
<td>EMP_VW</td>
</tr>
<tr>
<td>2</td>
<td>PROCEDURE</td>
<td>ADD_EMP</td>
</tr>
<tr>
<td>1</td>
<td>PROCEDURE</td>
<td>QUERY_EMP</td>
</tr>
</tbody>
</table>

4 rows returned in 0.29 seconds

In this example, `ADD_EMP` is directly dependent on `EMP_VW`, which in turn is directly dependent on `EMPLOYEES` (look at the `nested_level` column).
Tell Me / Show Me

Another Scenario of Local Dependencies

EMPLOYEES table

<table>
<thead>
<tr>
<th>EMPLOYEE_ID</th>
<th>LAST_NAME</th>
<th>JOB_ID</th>
<th>SALARY</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>King</td>
<td>AD_PRES</td>
<td>24000</td>
</tr>
<tr>
<td>101</td>
<td>Kochhar</td>
<td>AD_VP</td>
<td>17000</td>
</tr>
<tr>
<td>102</td>
<td>De Haan</td>
<td>AD_VP</td>
<td>17000</td>
</tr>
<tr>
<td>103</td>
<td>Hunold</td>
<td>IT_PROG</td>
<td>9000</td>
</tr>
<tr>
<td>104</td>
<td>Ernst</td>
<td>IT_PROG</td>
<td>6000</td>
</tr>
</tbody>
</table>

...
Tell Me / Show Me

A Third Scenario of Local Naming Dependencies

QUERY_EMP procedure

EMPLOYEES table

EMPLOYEES public synonym

<table>
<thead>
<tr>
<th>EMPLOYEE_ID</th>
<th>LAST_NAME</th>
<th>JOB_ID</th>
<th>SALARY</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>King</td>
<td>AD_PRES</td>
<td>24000</td>
</tr>
<tr>
<td>101</td>
<td>Kochhar</td>
<td>AD_VP</td>
<td>17000</td>
</tr>
<tr>
<td>102</td>
<td>De Haan</td>
<td>AD_VP</td>
<td>17000</td>
</tr>
<tr>
<td>103</td>
<td>Hunold</td>
<td>IT_PROG</td>
<td>9000</td>
</tr>
<tr>
<td>104</td>
<td>Ernst</td>
<td>IT_PROG</td>
<td>6000</td>
</tr>
</tbody>
</table>
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]];```

Tell Me / Show Me

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
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
Tell Me / Show Me

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
Tell Me / Show Me

Packages and Dependencies

Stand-alone procedure

Valid

Package specification

Procedure A declaration

Valid

Package body

Definition changed

Procedure A definition
Tell Me / Show Me

Packages and Dependencies (continued)

- Package specification
  - Procedure A declaration: Valid

- Package body
  - Procedure A definition: Invalid
  - Stand-alone procedure: Definition changed

Copyright © 2007, Oracle. All rights reserved.
In this lesson, you learned to:

• Describe the implications of procedural dependencies
• Contrast dependent objects and referenced objects
• View dependency information in the dictionary views
• Use the UTLDTREE script to create the objects required to display dependencies
• Use the IDEPTREE and DEPTREE views to display dependencies
• Describe when automatic recompilation occurs
• List how to minimize dependency failures
Try It / Solve It

The exercises in this lesson cover the following topics:

• Describing the implications of procedural dependencies
• Describing dependent objects and referenced objects
• Viewing dependency information in the dictionary views
• Using the UTLDTREE script
• Using the IDEPTREE and DEPTREE views
• Listing how to minimize dependency failures