Dependencies

Oracle Academy Study Materials

Understanding Dependencies

- Some objects reference other objects as part of their definitions.
- Example: a stored procedure could contain a SELECT statement that selects columns from a table.
- The stored procedure is called a **dependent object**, whereas the table is called a **referenced object**.
- A PL/SQL subprogram can execute correctly only if the objects that it references exist and are valid.
- These objects can be tables, views, other PL/SQL subprograms, and other kinds of database objects.

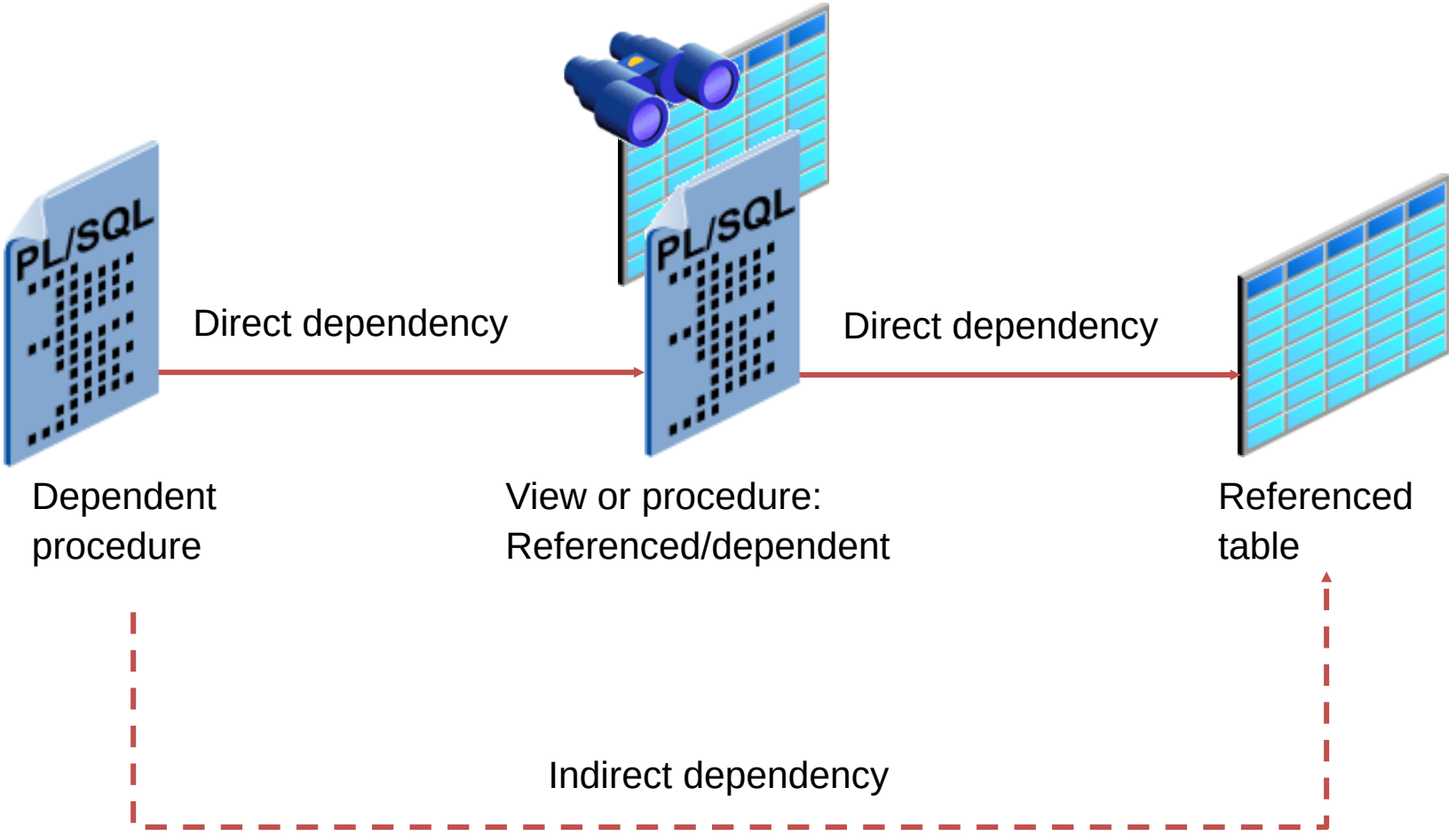
Understanding Dependencies

- If you alter the definition of a referenced object, dependent objects might not continue to work properly.
- Example: if the table definition is changed, the procedure might 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.

Overview of Schema Object Dependencies

Object Type	Can Be Dependent or Referenced
Package body	Dependent only
Package specification	Both
Sequence	Referenced only
Subprogram	Both
Synonym	Both
Table	Both
Trigger	Both
User-defined object	Both
User-defined collection	Both
View	Both

Understanding Dependencies



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 tries to recompile it.

- If the referenced object is in a different database, the dependency is called a **remote dependency**.

Querying an Object's Status

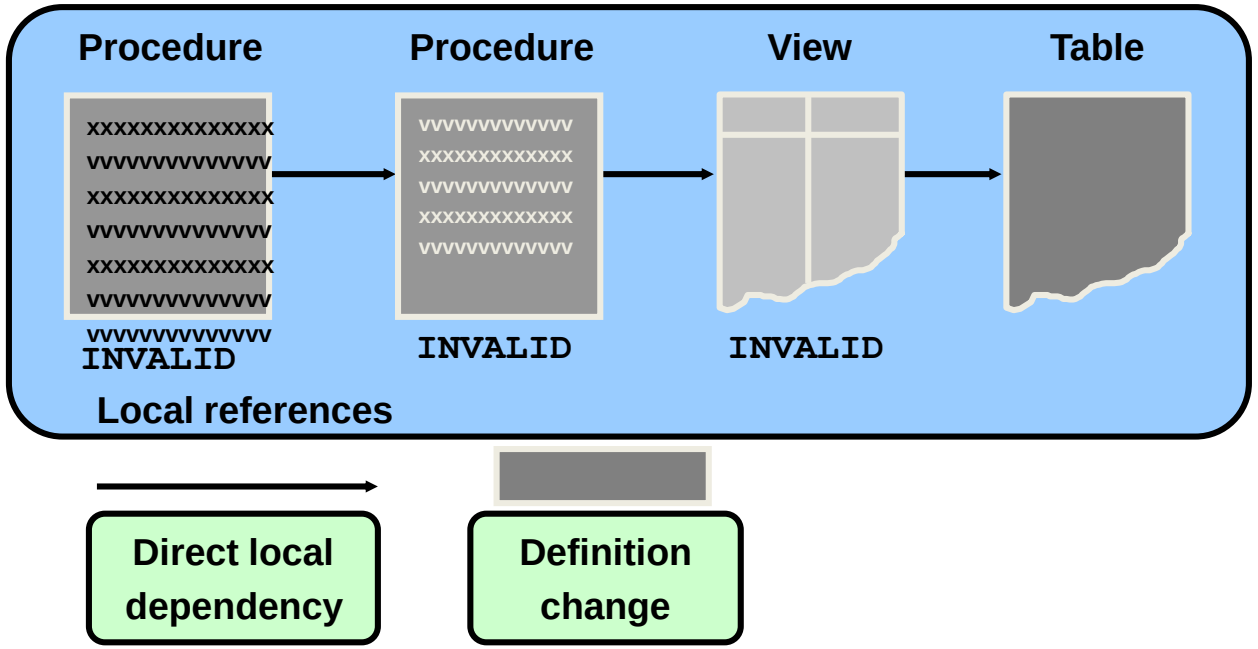
– Every database object has one of the following status values:

Status	Description
`VALID`	The object was successfully compiled, using the current definition in the data dictionary.
`COMPILED WITH ERRORS`	The most recent attempt to compile the object produced errors.
`INVALID`	The object is marked invalid because an object that it references has changed. (Only a dependent object can be invalid.)
`UNAUTHORIZED`	An access privilege on a referenced object was revoked. (Only a dependent object can be unauthorized.)

Understanding Dependencies

- Direct dependents are invalidated only by changes to the referenced object that affect them.
- Indirect dependents can be invalidated by changes to the reference object that do not affect them.

– The Oracle server implicitly attempts to recompile any INVALID object when the object is next called.



Understanding Dependencies

BUT:

- 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.

Displaying Direct and Indirect Dependencies

– Direct dependencies can be viewed through the USER_DEPENDENCIES and ALL_DEPENDENCIES views.

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

– Indirect dependencies can be viewed by utldtree.sql that creates four objects:

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

Recompiling a PL/SQL Program Unit

Recompilation is either:

- handled automatically through implicit run-time recompilation, or
- 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]];
```

- 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.

Recompiling a PL/SQL Program Unit

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.

Recompiling a PL/SQL Program Unit

Successful Recompilation

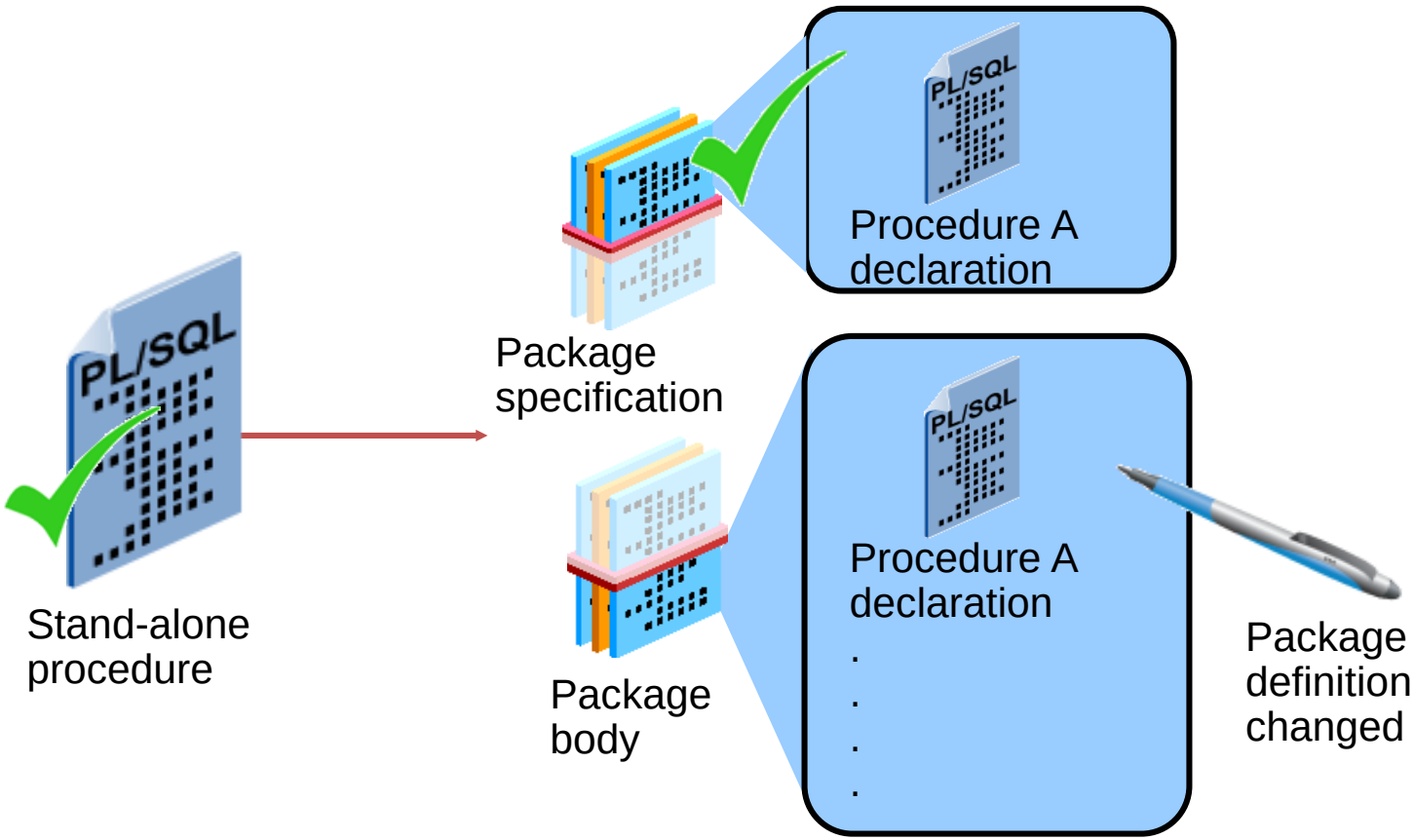
- Recompiling dependent procedures and functions is successful if:
- New columns are added to a referenced table.
- All INSERT statements include a column list.
- No new column is defined as NOT NULL.
- The data type of referenced columns has not changed.
- A private table is dropped, but a public table that has the same name and structure exists.
- The PL/SQL body of a referenced procedure has been modified and recompiled successfully.

Recompiling a PL/SQL Program Unit

Minimize dependency failures by:

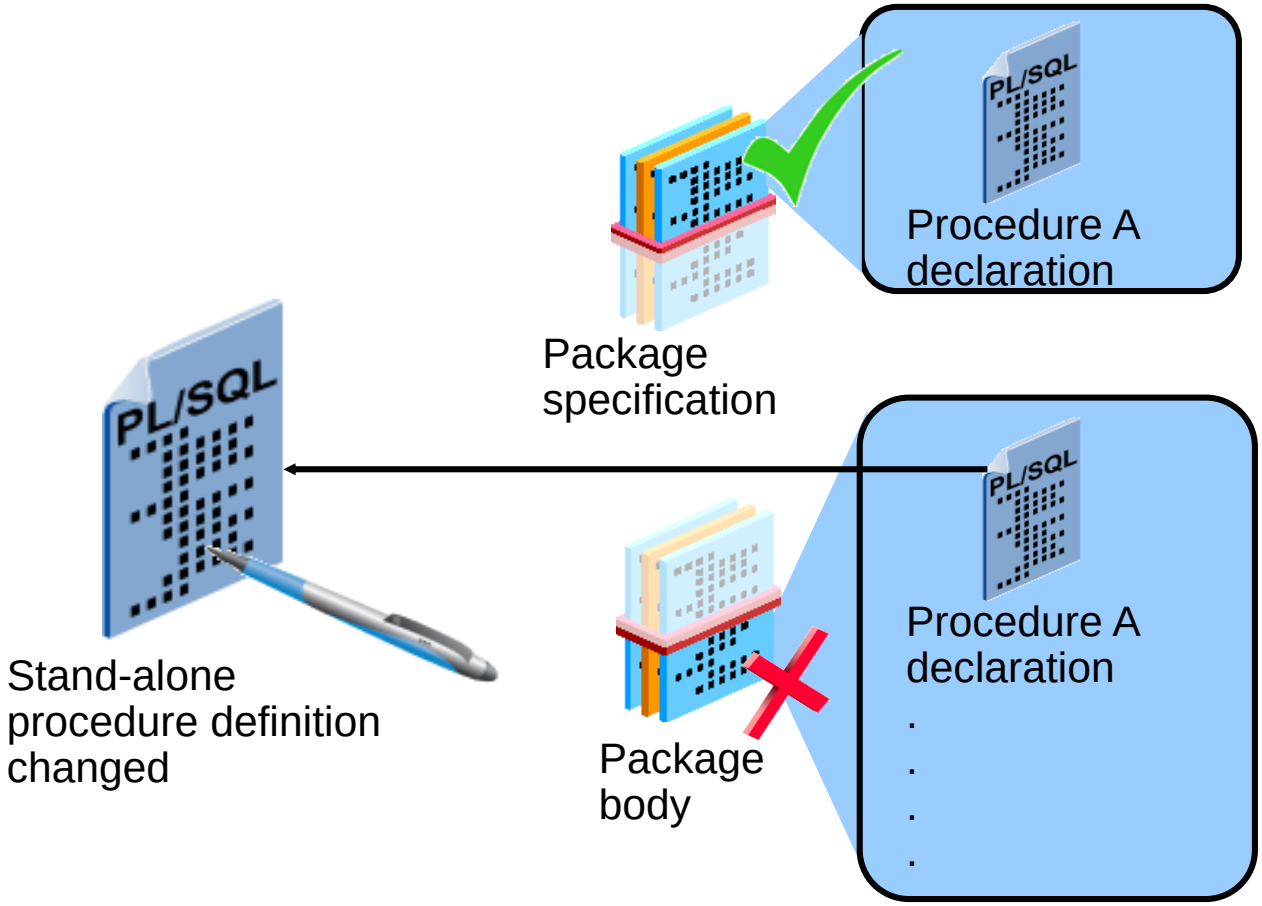
- Declaring records with the %ROWTYPE attribute
- Declaring variables with the %TYPE attribute
- Querying with the SELECT * notation
- Including a column list with INSERT statements

Packages and Dependencies



- 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



- 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.

Hiding a Source Code

Oracle Academy Study Materials

Hiding a Source Code

- You created a clever PL/SQL package. Now you may want other users to execute it, but you don't always want them to be able to see the details of the package's source code.
- when you create a PL/SQL program –procedure, function or package –the source code is loaded into the Data Dictionary, and you can see it using the Data Dictionary view USER_SOURCE:

```
CREATE OR REPLACE PROCEDURE mycleverproc
  (p_param1 IN   NUMBER, p_param2 OUT NUMBER)
IS BEGIN
  ... /* some clever but private code here */
END mycleverproc;
```

```
SELECT TEXT FROM USER_SOURCE
  WHERE TYPE = 'PROCEDURE' AND NAME = 'MYCLEVERPROC'
  ORDER BY LINE;
```

Hiding a Source Code

- Anyone who has EXECUTE privilege on a procedure or function can see your source code in ALL_SOURCE.
- We can hide the source code by converting it into a set of cryptic codes before we compile the subprogram.
- Hiding the source code is called **obfuscation**, and converting the source code to cryptic codes is called **wrapping the code**.
- When we compile the subprogram, only the wrapped code (the cryptic codes) is loaded into the Data Dictionary.
- Programs known as obfuscators transform human-readable code into obfuscated code using various techniques.
- Obfuscating code is typically done to manage risks that stem from unauthorized access to source code.

Hiding a Source Code

Benefits of Obfuscating:

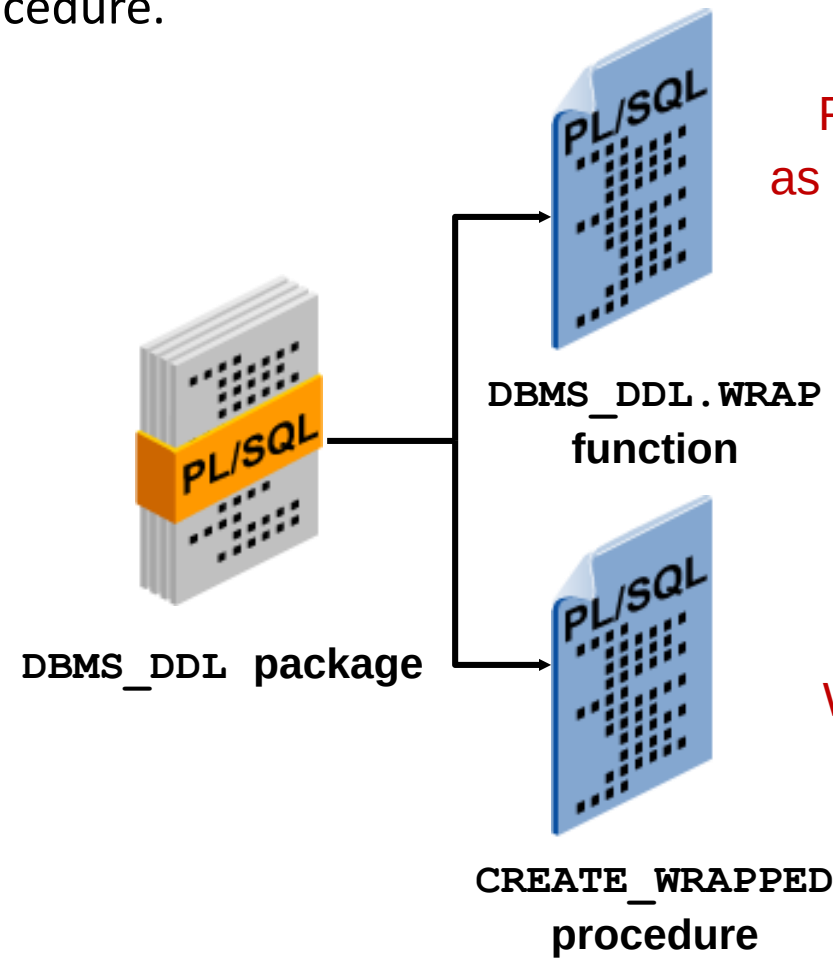
- It prevents others from seeing your source code.
- Your source code is not visible through the USER_SOURCE, ALL_SOURCE, or DBA_SOURCE data dictionary views.
- SQL*Plus can process the obfuscated source files.
- The Import and Export utilities accept wrapped files.

Wrapping can be done with:

- the wrap utility
  - The wrap utility is run from the command line and it processes an input SQL file, such as a SQL*Plus installation script.
- DBMS_DDL subprograms
  - The DBMS_DDL subprograms wrap a single PL/SQL unit, such as a single CREATE PROCEDURE command, that has been generated dynamically.

Hiding a Source Code

– The DBMS_DDL package contains the WRAP function and the CREATE_WRAPPED procedure.



Provides the same functionality as the `CREATE_WRAPPED` procedure but allows for larger inputs

Wraps the text and creates the PL/SQL unit

Hiding a Source Code

Using the DBMS_DDL.CREATE_WRAPPED Procedure:

- We must pass the complete code of our subprogram as a single IN argument with data type VARCHAR2.
- A PL/SQL VARCHAR2 variable has a maximum size of 32,767 characters, so this is the maximum size of our source code.

```
BEGIN
  DBMS_DDL.CREATE_WRAPPED
  ('CREATE OR REPLACE PROCEDURE mycleverproc
   (p_param1 IN   NUMBER, p_param2 OUT NUMBER)
  IS BEGIN
    ... /* some clever but private code here */
  END mycleverproc;');
END;
```

Hiding a Source Code

Using the DBMS_DDL.CREATE_WRAPPED Procedure:

– If the source code is long, it may be easier to assign it line by line to a VARCHAR2 variable and pass the variable as an actual parameter to the CREATE_WRAPPED procedure:

```
DECLARE
  v_code  VARCHAR2(32767);
BEGIN
  v_code := 'CREATE OR REPLACE FUNCTION myclevererfunc '
    || '(p_param1 IN NUMBER) ' || 'RETURN NUMBER IS BEGIN '
    || '... /* some even cleverer but private code
      */ '
    || 'END myclevererfunc;';
  DBMS_DDL.CREATE_WRAPPED(v_code);
END;
```

Hiding a Source Code

- Wrapping Package Code
- You can wrap PL/SQL package body code just like procedures and functions:

```
BEGIN
  DBMS_DDL.CREATE_WRAPPED
  ('CREATE OR REPLACE PACKAGE BODY mycleverpack
   ...
   END mycleverpack;');
END;
```

- You can also try to wrap the package specification.
- You won't get an error, but the specification will not be obfuscated
  - Other developers need to see information in the specification in order to use the package.

Hiding a Source Code

Using the PL/SQL Wrapper Utility

To use the WRAP utility program, you must log into the operating system of your database server computer.

There are three steps:

1. Create a text file containing your complete unwrapped source code.
2. Execute WRAP to create a second text file containing the wrapped code.
3. Connect to the database and execute the wrapped text file as a script to compile the wrapped code into the Data Dictionary.

Hiding a Source Code

1. Create a text file containing your complete unwrapped source code.

   Use a text editor to create a file containing your complete source code, starting with CREATE OR REPLACE … and ending with END …; Let's suppose the text file is called mysourcecode.sql.

2. Execute WRAP to create a second text file containing the wrapped code.

```
C:> WRAP INAME=mysourcecode.sql
```

   This creates the wrapped code in a second file called mysourcecode.plb .

   You can give your .plb file a different name:

```
C:> WRAP INAME=mysourcecode.sql ONAME=mywrappedcode.plb
```

3. Connect to the database and execute the wrapped text file as a script to compile the wrapped code into the Data Dictionary.

Hiding a Source Code

Comparing the Two Methods for Wrapping Code:

Functionality	DBMS_DDL	Wrap Utility
Code obfuscation	Yes	Yes
Dynamic Obfuscation	Yes	No
Obfuscate multiple programs at a time	No	Yes

- For very large PL/SQL programs where the source code is more than 32,767 bytes, you must use the WRAP utility.
- For smaller programs, DBMS_DDL.CREATE_WRAPPED is easier because you don't need to log on to the database server machine, and everything is done in a single step.

Hiding a Source Code

Guidelines for Wrapping:

- You must wrap only the package body, not the package specification.
- The wrapper can detect syntactic errors but cannot detect semantic errors.
- The output file should not be edited. You maintain the original source code and wrap again as required.
- To ensure that all the important parts of your source code are obfuscated, view the wrapped file in a text editor before distributing it.