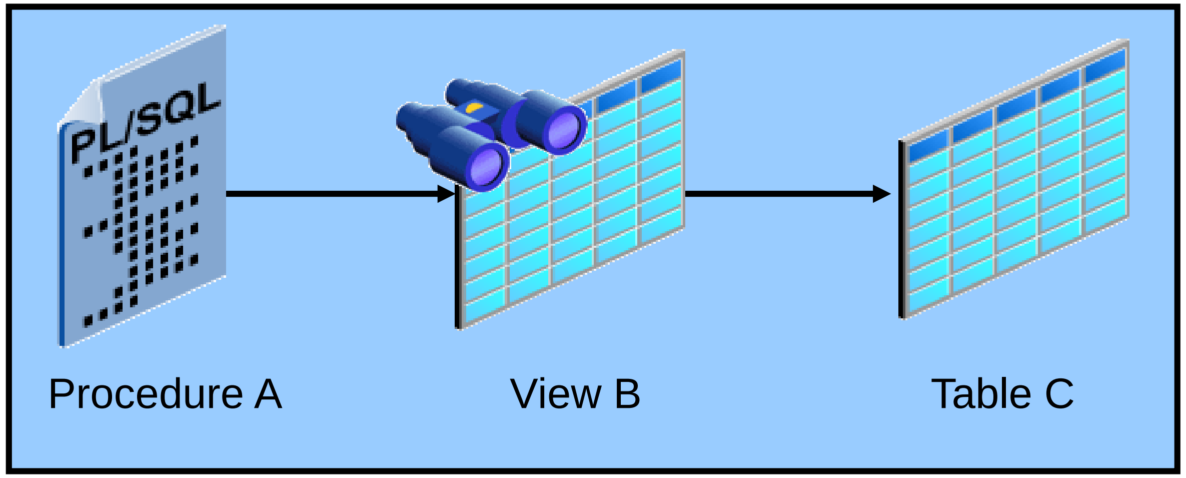
Managing Dependencies



```
create or replace view v_emp_copy
as
select * from emp_copy;
```

This is Dependent object

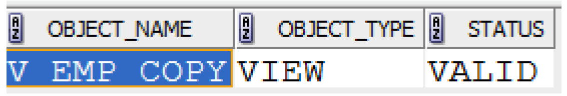
This is referenced object

1

If the definition of object A references object B then :
A is dependent object , b is referenced object

2

```
select object_name,object_type, status
from user_objects
where object_name='V_EMP_COPY';
```



OBJECT_NAME	OBJECT_TYPE	STATUS
V_EMP_COPY	VIEW	VALID

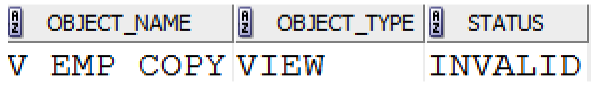
3

```
drop table emp_copy;
```

The view v_emp_copy reading from this table

4

```
select object_name,object_type, status
from user_objects
where object_name='V_EMP_COPY';
```



OBJECT_NAME	OBJECT_TYPE	STATUS
V_EMP_COPY	VIEW	INVALID

Overview of Schema Object Dependencies

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

IF you alter the definition of referenced object
Then dependent object may or may not work

```
create or replace view v_dept_copy
as
select department_id, department_name
from dept_copy;
```

```
ALTER TABLE dept_copy
ADD ( DSIZE NUMBER);
```

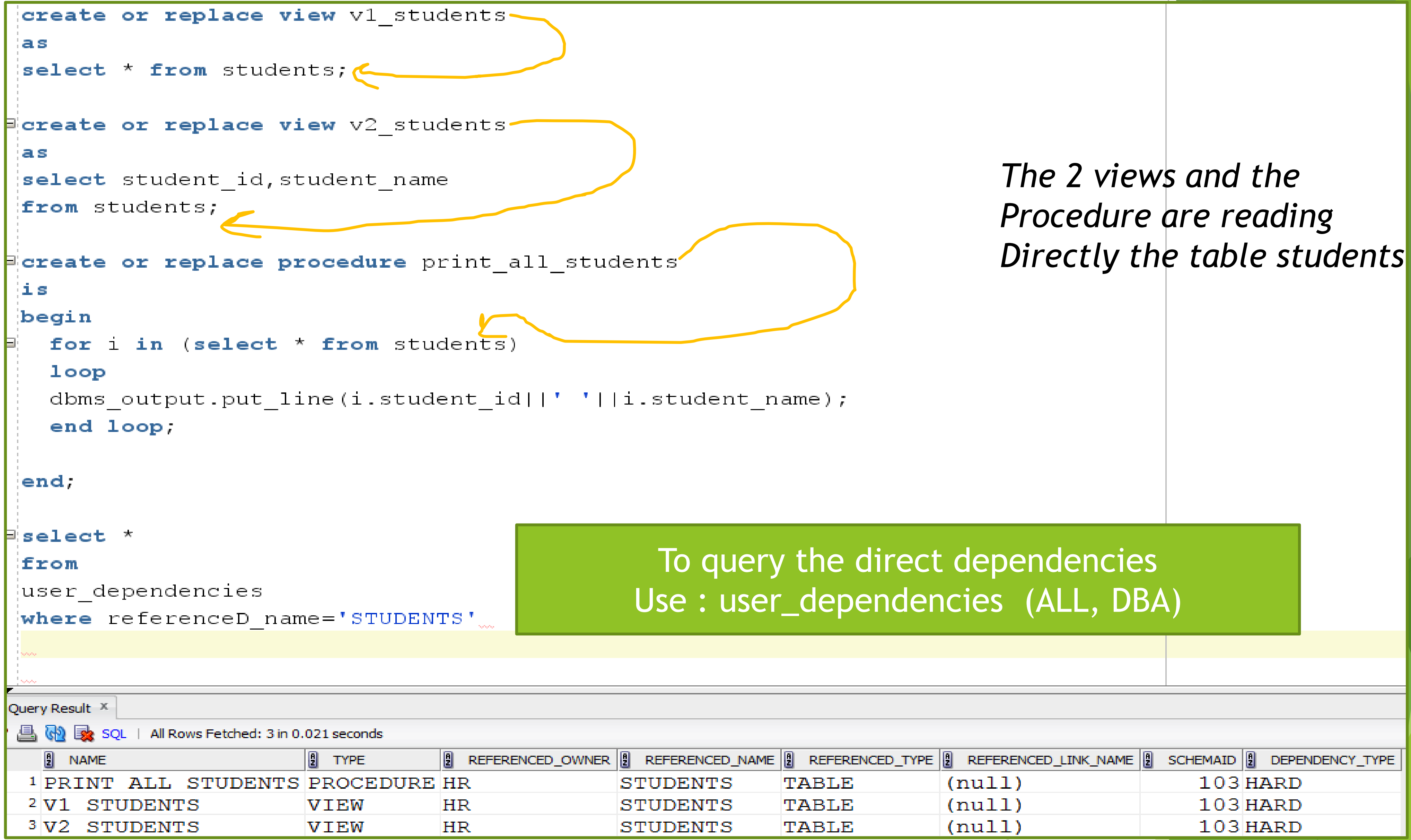
Here the changes will not affect
V_dept_copy still valid

this start from 11g

```
alter table dept_copy
drop column department_name;
```

Here the changes will affect
V_dept_copy will be invalid

Direct Local Dependencies *Example: Students Table*



```
create or replace view v1_students
as
select * from students;

create or replace view v2_students
as
select student_id,student_name
from students;

create or replace procedure print_all_students
is
begin
  for i in (select * from students)
  loop
  dbms_output.put_line(i.student_id||' '||i.student_name);
  end loop;

end;

select *
from
user_dependencies
where referenceD_name='STUDENTS'
```

Query Result

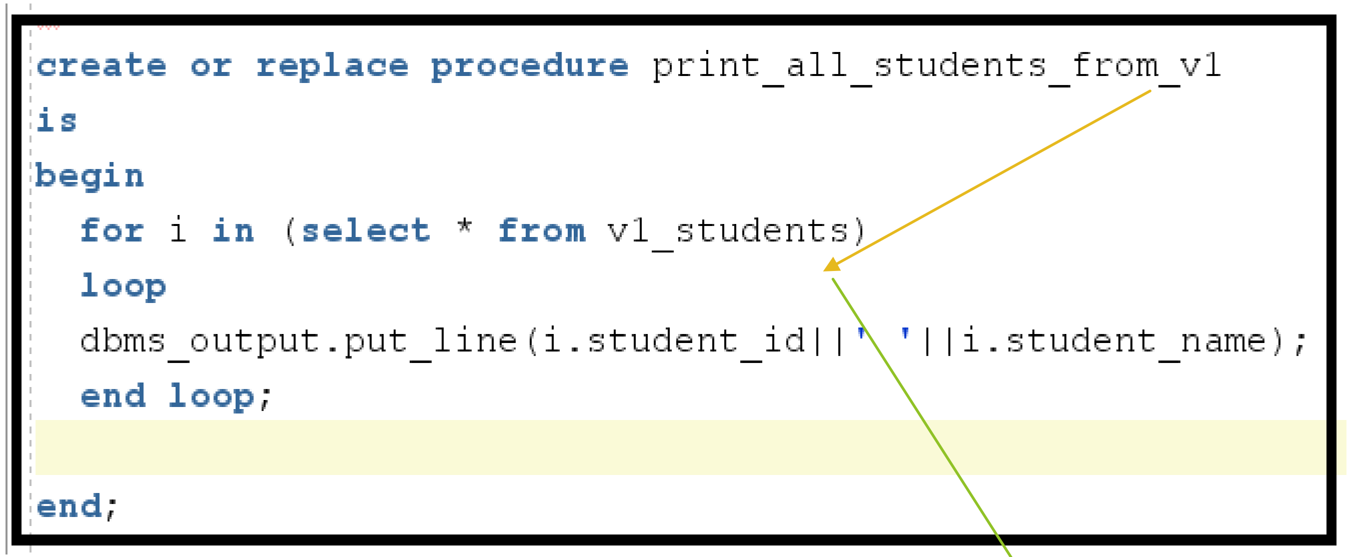
SQL | All Rows Fetched: 3 in 0.021 seconds

	NAME	TYPE	REFERENCED_OWNER	REFERENCED_NAME	REFERENCED_TYPE	REFERENCED_LINK_NAME	SCHEMAID	DEPENDENCY_TYPE
1	PRINT_ALL_STUDENTS	PROCEDURE	HR	STUDENTS	TABLE	(null)	103	HARD
2	V1_STUDENTS	VIEW	HR	STUDENTS	TABLE	(null)	103	HARD
3	V2_STUDENTS	VIEW	HR	STUDENTS	TABLE	(null)	103	HARD

The 2 views and the Procedure are reading Directly the table students

To query the direct dependencies
Use : user_dependencies (ALL, DBA)

Indirect Local Dependencies *Example: Students Table*

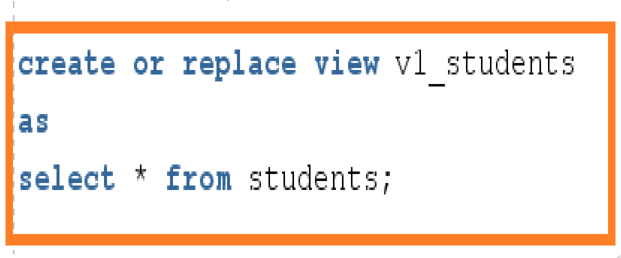


```
create or replace procedure print_all_students_from_v1
is
begin
   for i in (select * from v1_students)
   loop
   dbms_output.put_line(i.student_id||' '||i.student_name);
   end loop;

end;
```

The procedure read from the table
But not direct (form the view)

A procedure or function can directly or indirectly reference

- Tables
- Views
- Sequences
- Procedures
- Functions
- Packaged procedures Or function



```
create or replace view v1_students
as
select * from students;
```

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. `$ORACLE_HOME/rdbms/admin`

2. Execute the `DEPTREE_FILL` procedure.

```
EXECUTE deptree_fill('TABLE', 'ORA62', 'EMPLOYEES')
                      Object type  object owner  object name
```

Displaying Dependencies Using the DEPTREE View

3.

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

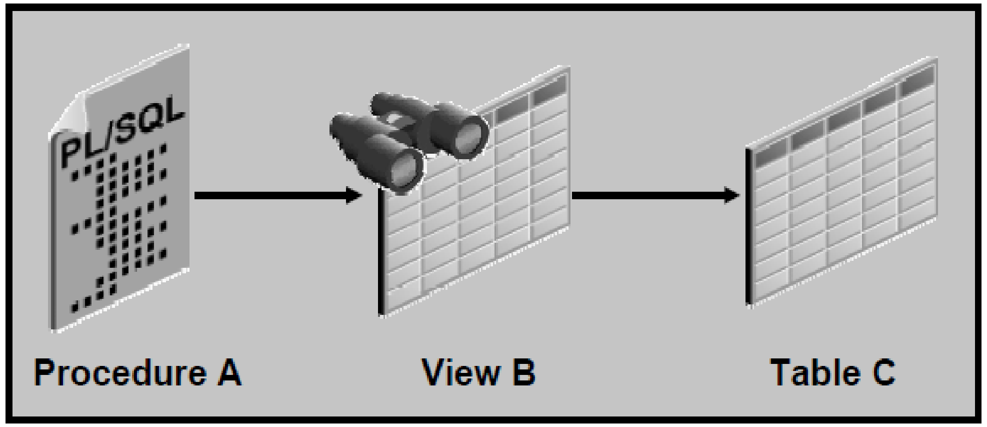
Querying an Object's Status

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.)

But in user_objects (DBA,ALL)
Only valid or invalid

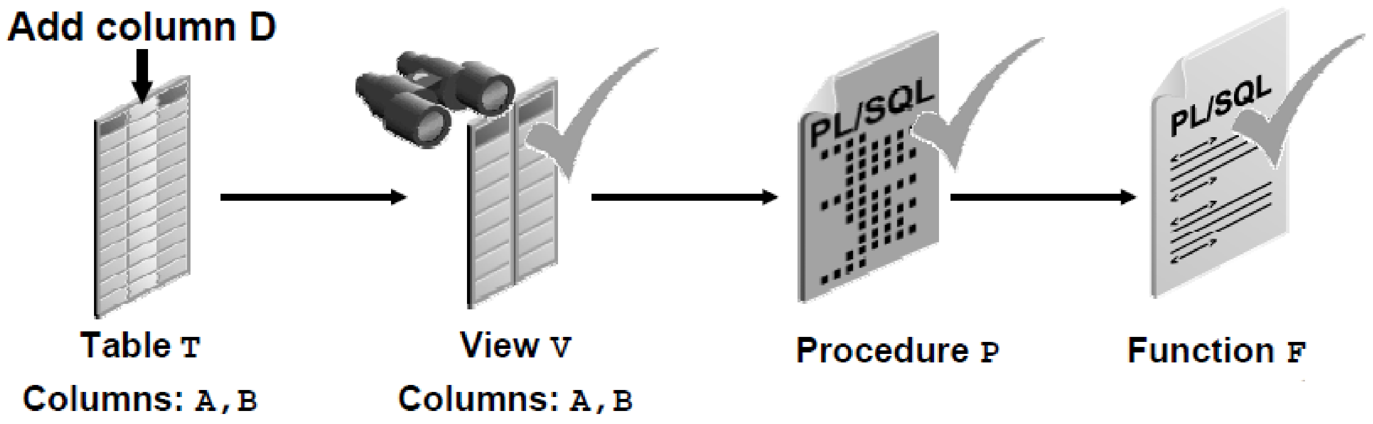
Invalidation of Dependent Objects



- Procedure A is a direct dependent of View B. View B is a direct dependent of Table C. Procedure A is an indirect dependent of Table C.
- 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.

More Precise Dependency Metadata in Oracle Database 11*g*

- Before 11*g*, adding column `D` to table `T` invalidated the dependent objects.
- Oracle Database 11*g* records additional, finer-grained dependency management:
  - Adding column `D` to table `T` does not impact view `V` and does not invalidate the dependent objects



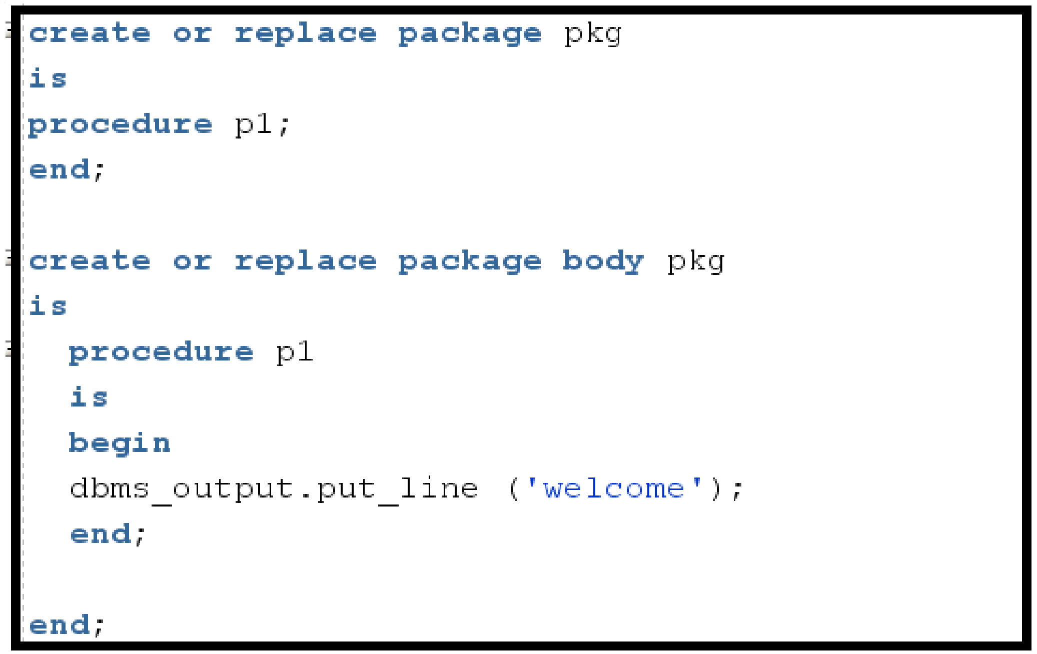
	View V	Procedure P	Function F
Bofore 11g	invalid	invalid	invalid
11g,12c	valid	valid	valid

Impact of Adding/ Altering column for the Referenced Table

Object	Adding column in referenced table	Alter column in referenced table
View	Validated	May or May not invalidate according for the logic of changes
Function	Invalid	May or May not invalidate according for the logic of changes
Procedure	May or May not according for the logic of changes	May or May not invalidate according for the logic of changes

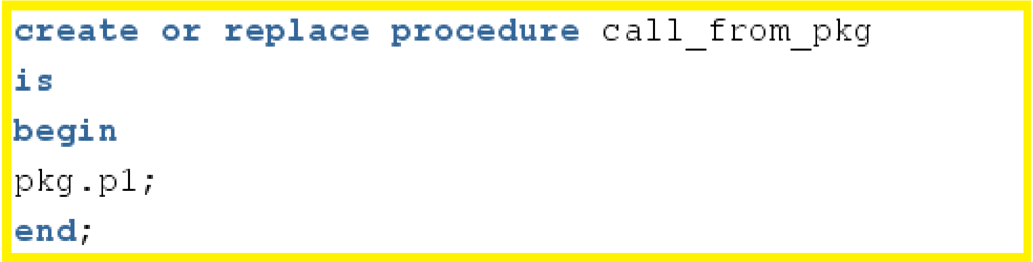
Packages and Dependencies

1



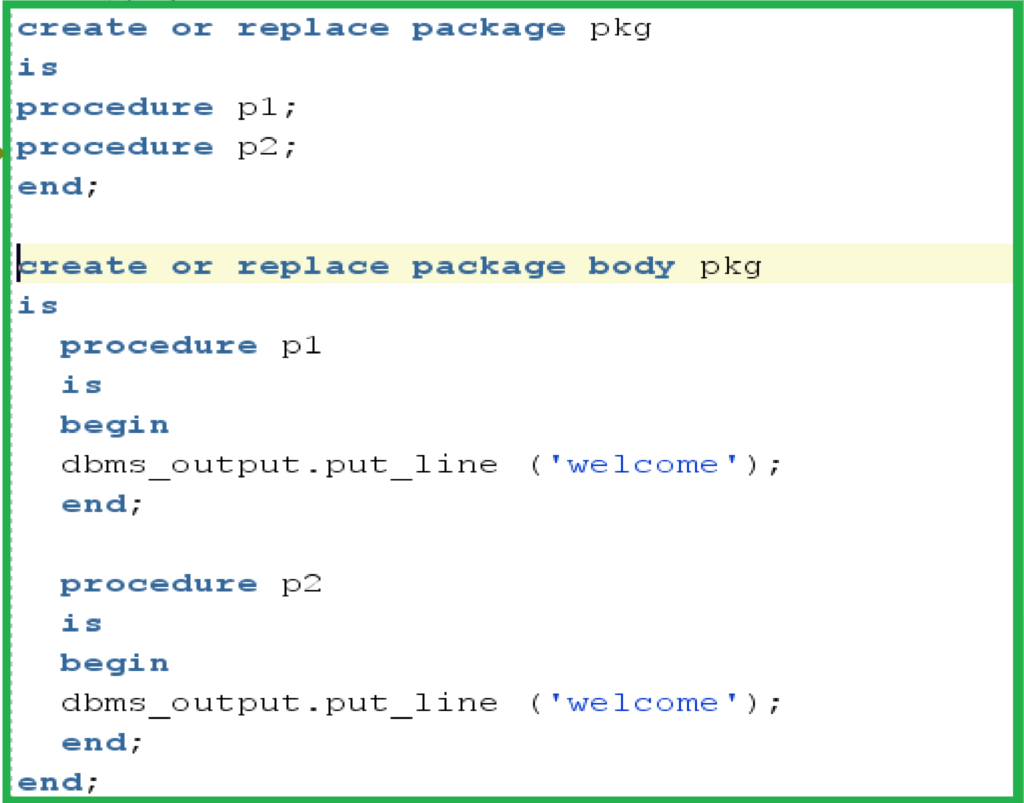
```
create or replace package pkg
is
procedure p1;
end;

create or replace package body pkg
is
  procedure p1
  is
  begin
  dbms_output.put_line ('welcome');
  end;

end;
```

2



```
create or replace procedure call_from_pkg
is
begin
pkg.p1;
end;
```

3



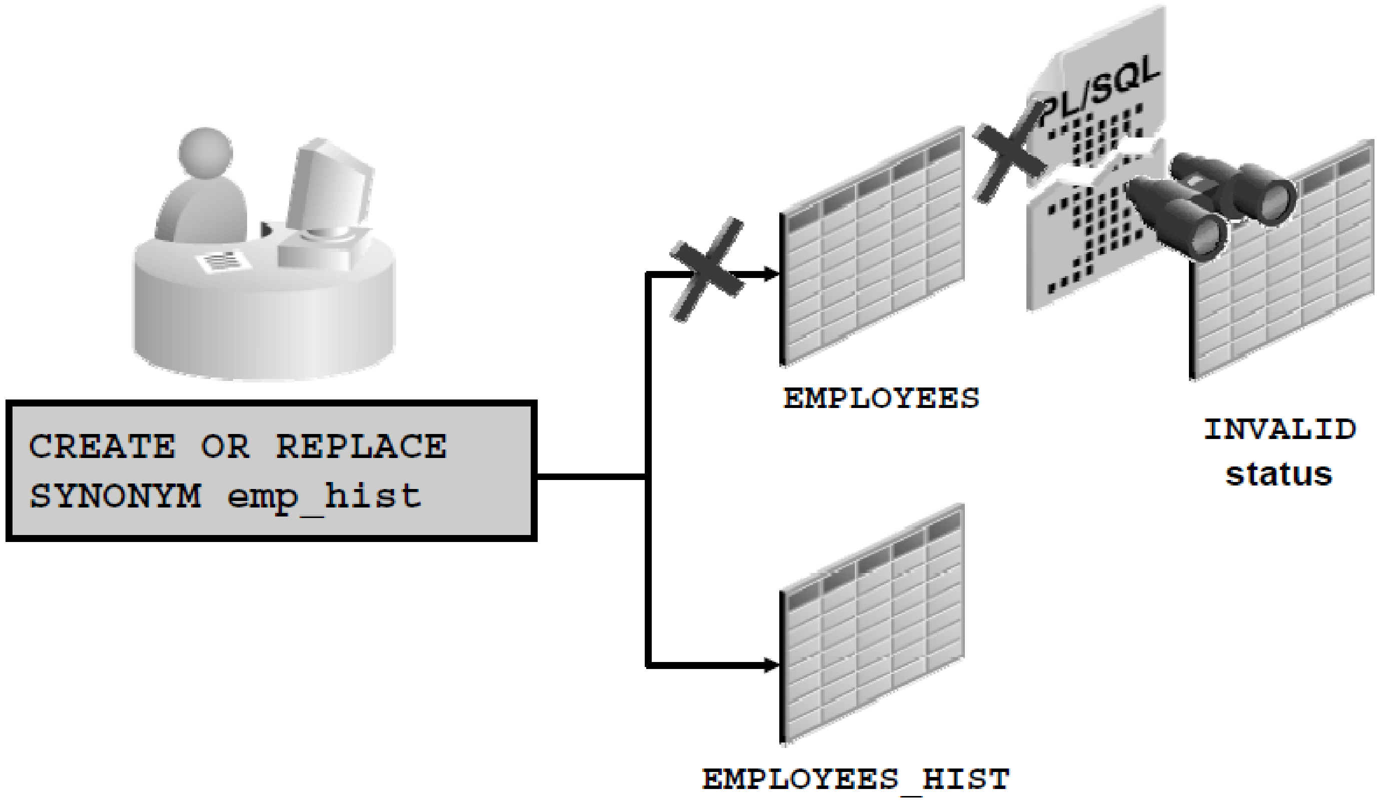
```
create or replace package pkg
is
procedure p1;
procedure p2;
end;

create or replace package body pkg
is
  procedure p1
  is
  begin
  dbms_output.put_line ('welcome');
  end;

  procedure p2
  is
  begin
  dbms_output.put_line ('welcome');
  end;
end;
```

4

select * from user_objects
where lower(object_name) ='call_from_pkg';

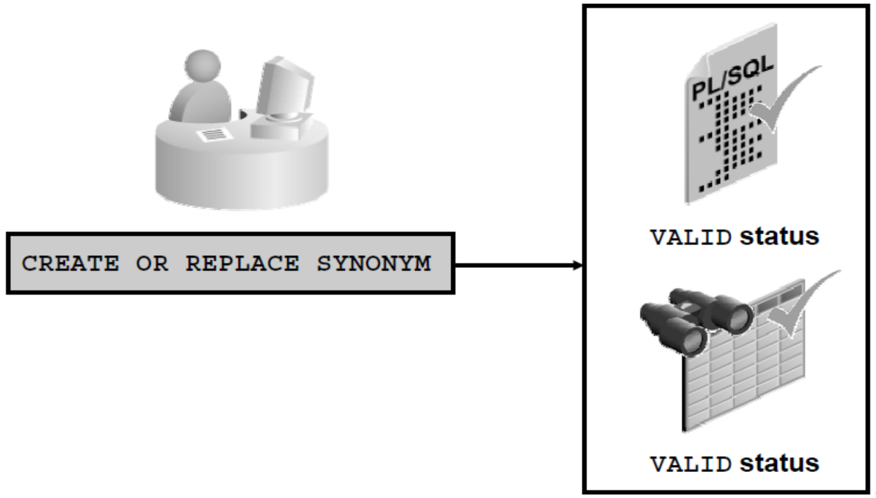
VALID

Impact of Redefining Synonyms Before Oracle Database 10*g*



Prior to Oracle Database 10*g*, when a synonym was re-created, the status of any dependent PL/SQL program unit was set to INVALID. If you did not recompile the PL/SQL program units manually, they would recompile automatically the next time they were invoked, causing runtime performance overhead.

Changes to Synonym Dependencies Starting with Oracle Database 10*g*



Now, Oracle Database 10*g* introduces automatic validation of PL/SQL program units, which determines whether program units will have to be invalidated. This is only the case in the following circumstances.

- ✍ The column order, column names, and column data types of the tables need to be 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 sub-partitions need to be identical.
- ✍ The tables are of the same organization type.
- ✍ Object type columns are of the exact same type.

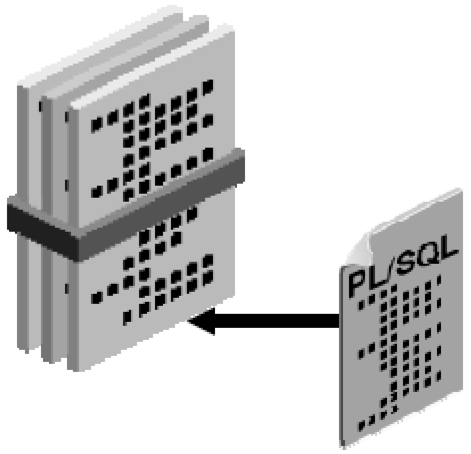
In the case of views, just like PL/SQL program units, even when the synonym definition is changed, the dependent views will not be validated. However, this only happens when certain conditions are met. All the conditions specified for PL/SQL units are relevant here also.

In addition, the following conditions must be met to keep the VALID status of the dependent views when you redefine a synonym:

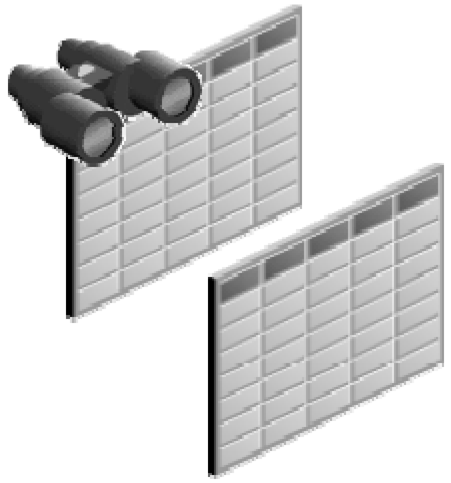
- ✍ Columns and order of columns defined for primary key and unique indexes, NOT NULL constraints, and primary key and unique constraints must be identical.
- ✍ The dependent view cannot have any referential constraints.

Guidelines for Reducing Invalidation

To reduce invalidation of dependent objects:



Add new items to the end of the package



Reference each table through a view

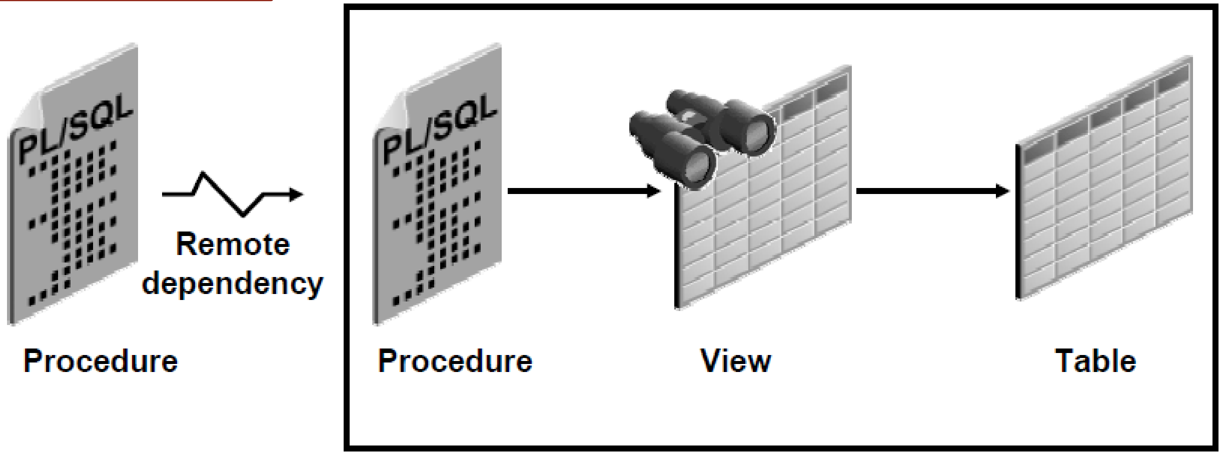
Object Revalidation

- An object that is not valid when it is referenced must be validated before it can be used.
- Validation occurs automatically when an object is referenced; it does not require explicit user action.
- If an object is not valid, its status is either `COMPILED WITH ERRORS`, `UNAUTHORIZED`, or `INVALID`.

Remote Dependencies

Local
hr@orclpdb

Remote
ict@newd

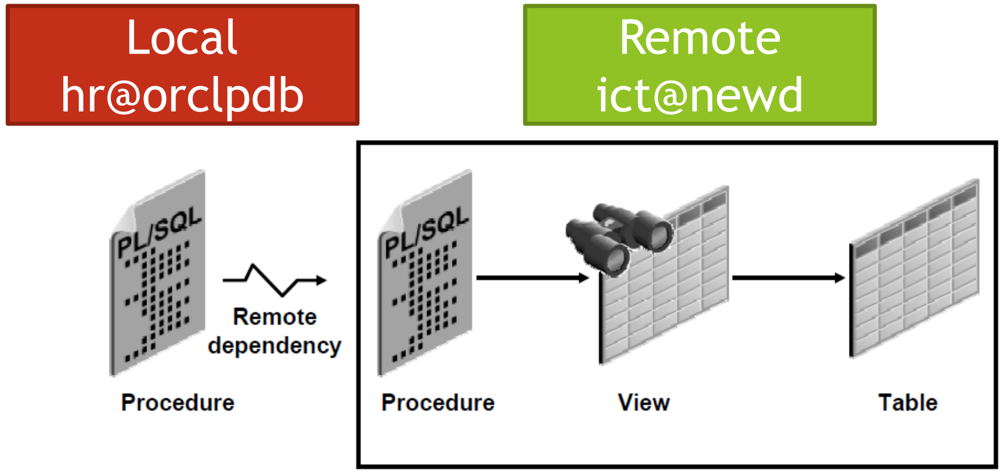


To read from remote DB, the DBA should create Database link.
So the DBA will create link in orclpdb that read newd database

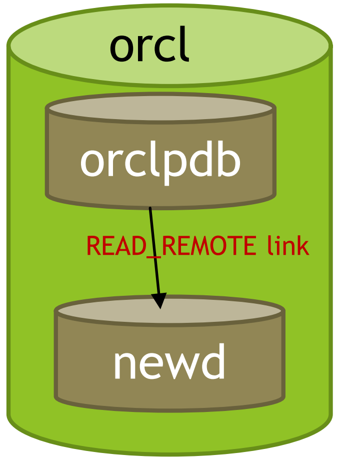
```
create public database link READ_REMOTE
connect to ict
identified by ict
using 'newd';
```

Now when user HR (orclpdb) need to query table from ict(newd): select * from table_name@database_link_name
Example : select * from students@READ_REMOTE;

Remote Dependencies



Local and remote references



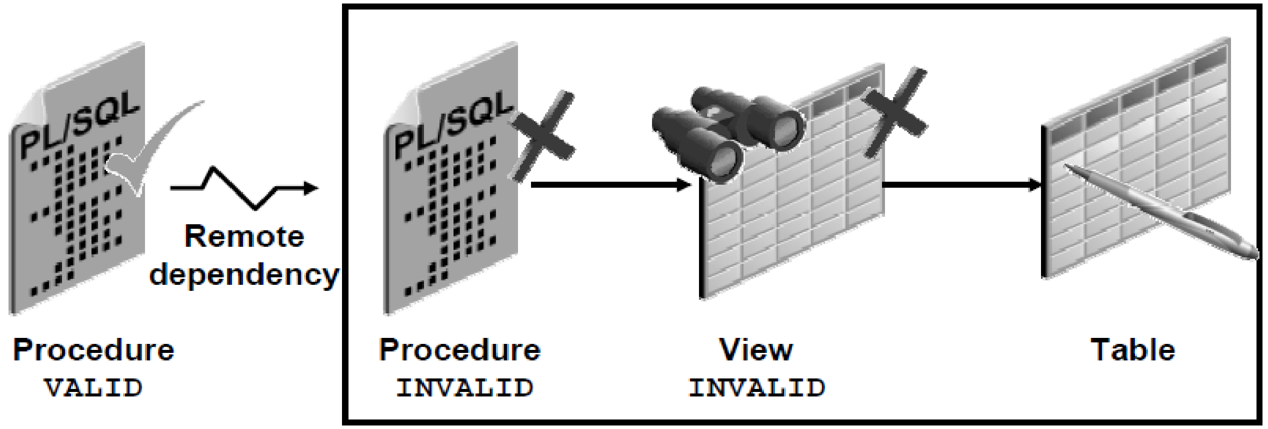
1- We will create new pluggable database called (newd)
we will use Database Configuration Assistant (DBCA)
2- We will create new user called (ICT) inside the new pluggable database (newd)

```
create user ict identified by ict;
grant create session to ict;
grant connect to ict;
grant create  table to ict;
grant create  procedure to ict;
grant create view to ict;
GRANT UNLIMITED TABLESPACE TO ict;
```

3- We will create database link from orclpdb to read the new pluggable database (newd)

```
create public database link READ_REMOTE
connect to ict
identified by ict using 'newd';
```

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 chosen by the user:

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

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 of 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.

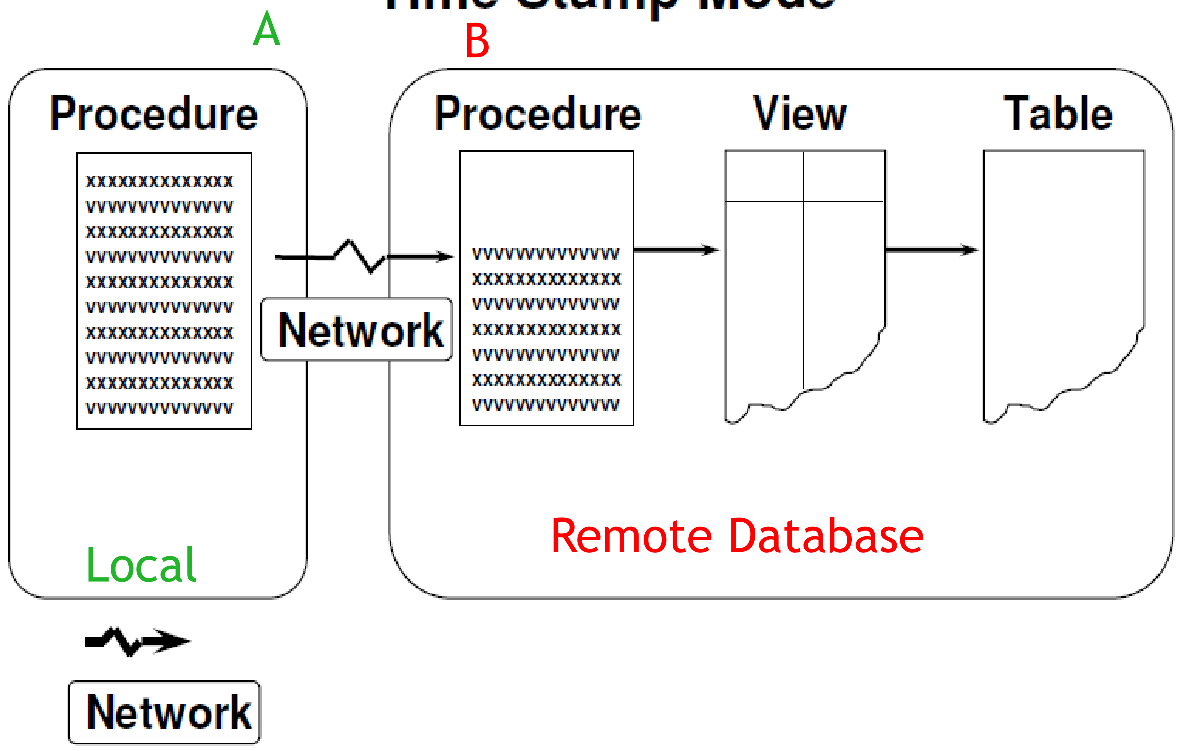
SIGNATURE Checking

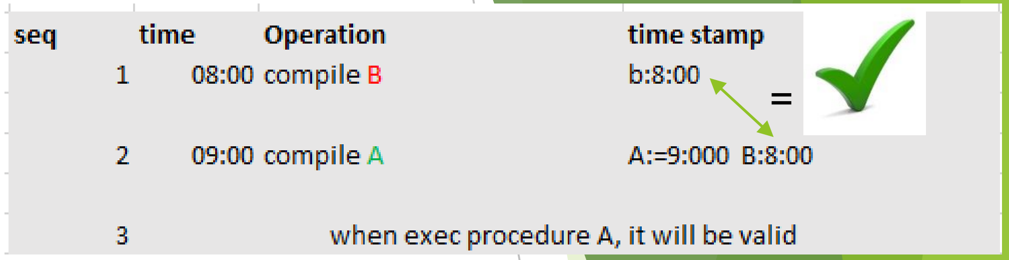
For each PL/SQL program unit, both the time stamp and the signature are recorded. The signature of a PL/SQL construct contains information about the following:

- 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

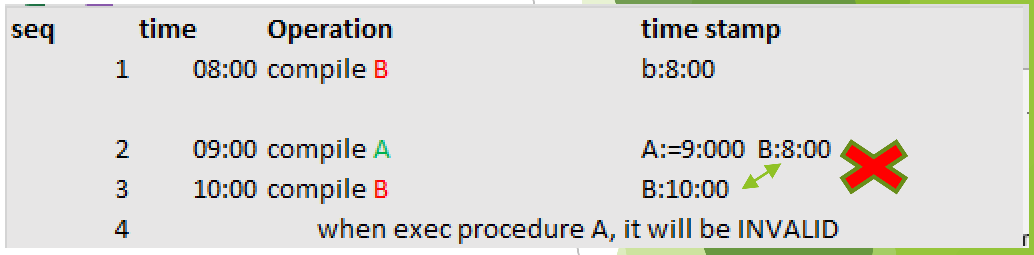
The recorded time stamp in the calling program unit is compared with the current time stamp in the called remote program unit. If the time stamps match, the call proceeds normally. If they do not match, the Remote Procedure Calls (RPC) layer performs a simple test to compare the signature to determine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

Remote Dependencies and Time Stamp Mode





seq	time	Operation	time stamp
1	08:00	compile B	b:8:00
2	09:00	compile A	A:=9:000 B:8:00
3		when exec procedure A, it will be valid	



seq	time	Operation	time stamp
1	08:00	compile B	b:8:00
2	09:00	compile A	A:=9:000 B:8:00
3	10:00	compile B	B:10:00
4		when exec procedure A, it will be INVALID	

Using Time Stamp Mode for Automatic Recompilation of Local and Remote Objects

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 of its dependent units are marked as invalid and must be recompiled before they can be run.

REMOTE_DEPENDENCIES_MODE Paramete

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
```

Setting the `REMOTE_DEPENDENCIES_MODE`

value	TIMESTAMP
	SIGNATURE

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 number of parameters
  - The datatype of the return value for a function
- 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.

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]];
```

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

Recompiling 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

▶ Thank You