18 Managing Dependencies

Understanding Schema Object Dependency - Oracle 19c Database Development Guide

*_DEPENDENCIES: READ ONLY

DEPTREE: READ ONLY

IDEPTREE: READ ONLY

*_OBJECTS: READ ONLY

Overview of Schema Object Dependency

- A dependency (in PL/SQL) is a reference from a stored program to some database object outside that program.
- Server-based PL/SQL programs can have dependencies on `tables, views, types, procedures, functions, sequences, synonyms, object types, package specifications, etc.` Program units are not, however, dependent on package bodies or object type bodies; these are the "hidden" implementations.
- Do not use the currently compiled version of a program if any of the objects on which it depends have changed since it was compiled.
- Some types of schema objects can reference other objects in their definitions, e.g. a view is defined by a query that references tables/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** (of B) and B is a **referenced object** (of A)
- If you alter the definition of a referenced object, dependent objects 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
- `CREATE` statements automatically update all dependencies
- Dynamic SQL statements do not create dependencies. For example, this statement does not create a dependency on tab1:

```
EXECUTE IMMEDIATE 'SELECT * FROM tab1'
```

Analyzing Dependencies with Data Dictionary Views, Querying Object Dependencies

- Display dependent and referenced objects with: `USER_, ALL_, DBA_ DEPENDENCIES`

- The static data dictionary views `USER_DEPENDENCIES`, `ALL_DEPENDENCIES`, and `DBA_DEPENDENCIES` describe dependencies between database objects

```
SELECT name, type, referenced_owner, referenced_name, referenced_type
FROM USER_DEPENDENCIES;
```

NAME	TYPE	REFERENCED_OWNER	REFERENCED_NAME	REFERENCED_TYPE
SHOWCARD	PROCEDURE	SYS	STANDARD	PACKAGE
SHOWCARD	PROCEDURE	SYS	SYS_STUB_FOR_PURITY_ANALYSIS	PACKAGE
MEDEWERKERS_VU	VIEW	BOEK	MEDEWERKERS	TABLE
STUDENTEN_VU	VIEW	TESTING	STUDENTEN_EXT	TABLE
SHOWCARD	PROCEDURE	TESTING	SHOW_CARD	PROCEDURE

- The database can only track dependencies of local stored objects written with static calls
- There are plenty of ways that you can create programs that do not appear in the `USER_DEPENDENCIES` view. These include external programs that embed SQL or PL/SQL, remote stored procedures or client-side tools that call local stored objects, and any programs that use dynamic SQL
- The utldtree.sql SQL script creates the view `DEPTREE`, which contains information on the object dependency tree, and the view `IDEPTREE`, a presorted, pretty-print version of `DEPTREE`

Object status

- **Valid** - The object was successfully compiled, using the current definition in the data dictionary
- **Invalid** - The object is marked invalid because an object that it references has changed. (Only a dependent object can be invalid.)
- **Compiled with errors** - The most recent attempt to compile the object produced errors
- **Unauthorized** - An access privilege on a referenced object was revoked. (Only a dependent object can be unauthorized.)
- > Note: The static data dictionary views `USER_OBJECTS`, `ALL_OBJECTS`, and `DBA_OBJECTS` do not distinguish between "Compiled with errors," "Invalid," and "Unauthorized"—they describe all of 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
- 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 C invalidates B, it invalidates A (and all other direct and indirect dependents of B). This is called **cascading invalidation**.
- With **coarse-grained invalidation**, a data definition language (DDL) statement that changes a referenced object invalidates all of its dependents.
- With **fine-grained invalidation**, a DDL statement that changes a referenced object invalidates only dependents for which either of these statements is true:
    - The dependent relies on the attribute of the referenced object that the DDL statement changed.

    - The compiled metadata of the dependent is no longer correct for the changed referenced object.
- For example, if view v selects columns c1 and c2 from table t, a DDL statement that changes only column c3 of t does not invalidate v
- The DDL statement `CREATE OR REPLACE` object has no effect under these conditions:
    - object is a PL/SQL object, the new PL/SQL source text is identical to the existing PL/SQL source text, and the PL/SQL compilation parameter settings stored with object are identical to those in the session environment
    - object is a synonym and the statement does not change the target object
- Table with all fine-grained and coarse-grained invalidation
- **Purity** refers to a set of rules for preventing side effects (such as unexpected data changes) when invoking PL/SQL functions within SQL queries. **Package purity** refers to the purity of the code in the package initialization block
- The **entry-point number** of a procedure or function is determined by its location in the PL/SQL package code. A procedure or function added to the end of a PL/SQL package is given a new entry-point number

Program units or objects may be invalidated if a change is made to the structure of the referenced object, one example is when a program unit uses `%ROWTYPE` and we modify or drop a column of the referenced table. Another simple example is as follows

```
create table emp(id number);
/
create view emp_vu as select * from emp;
/
drop table emp;
/
select object_name, object_type, status
from user_objects;
```

OBJECT_NAME	OBJECT_TYPE	STATUS
MEDEWERKERS_VU	VIEW	VALID
STUDENTEN_VU	VIEW	VALID
SHOWCARD	PROCEDURE	VALID
EMP_VU	VIEW	INVALID

> Note: Even if package body gets invalidated, as long as the specification doesn't change, program units that depend on the package will not be invalidated.

Session State and Referenced Packages

- Each session that references a package construct has its own instantiation of that package, including a persistent state of any public and private variables, cursors, and constants
- All of a session's package instantiations, including state, can be lost if any of the session's instantiated packages are subsequently invalidated and revalidated

Security Authorization

- When a data manipulation language (DML) object or system privilege is granted to, or revoked from, a user or PUBLIC, Oracle Database invalidates all the owner's dependent objects, to verify that an owner of a dependent object continues to have the necessary privileges for all referenced objects

Guidelines for Reducing Invalidation

- To reduce invalidation of dependent objects, follow these guidelines:
  - Add Items to End of Package
    - When adding items to a package, add them to the end of the package. This preserves the entry point numbers of existing top-level package items, preventing their invalidation
  - Reference Each Table Through a View
    - Reference tables indirectly, using views, enabling you to:
      - 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
    - The statement `CREATE OR REPLACE VIEW` does not invalidate an existing view or its dependents if the new `ROWTYPE` matches the old `ROWTYPE`

Revalidation of Objects

- 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
- Revalidation of Objects that Compiled with Errors:
  - 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
- Revalidation of Unauthorized Objects:
  - 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
- Revalidation of Invalid SQL Objects:
  - The SQL compiler recompiles the invalid object. If the object recompiles without errors, it is revalidated; otherwise, it remains invalid
- Revalidation of Invalid PL/SQL Objects:
  - 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

Recompilation

- No PL/SQL program marked as `INVALID` will run until a successful recompilation changes its status to `VALID`.
- Recompilation can happen in one of three ways:
  - Automatic runtime recompilation (when program unit is called)

    - `ALTER ... COMPILE` recompilation
    - Schema-level recompilation:
        - `utlip.sql`
        - `utlrp.sql`
        - `utlrcmp.sql`
        - `DBMS_UTILITY.COMPILE_SCHEMA`
        - `UTL_RECOMP`

Local Dependency Management

- **Local dependency management** occurs when Oracle Database manages dependencies among the objects in a single database. For example, a statement in a procedure can reference a table in the same database.

Remote Dependency Management

- **Remote dependency management** occurs when Oracle Database manages dependencies in distributed environments across a network. For example, an Oracle Forms trigger can depend on a schema object in the database. In a distributed database, a local view can reference a remote table
- Oracle Database also manages distributed database dependencies. For example, an Oracle Forms application might contain a trigger that references a table. The database system must account for dependencies among such objects
- Oracle Database uses different mechanisms to manage remote dependencies, depending on the objects involved

Dependencies Among Local and Remote Database Procedures

- Dependencies among stored procedures (including functions, packages, and triggers) in a distributed database system are managed using either time-stamp checking or signature checking
- The dynamic initialization parameter `REMOTE_DEPENDENCIES_MODE` determines whether time stamps or signatures govern remote dependencies

Dependencies Among Other Remote Objects

- Oracle Database does not manage dependencies among remote schema objects other than local-procedure-to-remote-procedure dependencies
- For example, assume that a local view is created and defined by a query that references a remote table. Also assume that a local procedure includes a SQL statement that references the same remote table. Later, the definition of the table is altered
- Therefore, the local view and procedure are never invalidated, even if the view or procedure is used after the table is altered, and even if the view or procedure now returns errors when used. In this case, the view or procedure must be altered manually so that errors are not returned. In such cases, lack of dependency management is preferable to unnecessary recompilations of dependent objects

Dependencies of Applications

- Code in database applications can reference objects in the connected database. For example, Oracle Call Interface (OCI) and precompiler applications can submit anonymous PL/SQL blocks. Triggers in Oracle Forms applications can reference a schema object

- Such applications are dependent on the schema objects they reference. Dependency management techniques vary, depending on the development environment. Oracle Database does not automatically track application dependencies

Remote Procedure Call (RPC) Dependency Management

- Remote procedure call (RPC) dependency management occurs when a local stored procedure calls a remote procedure in a distributed database system
- The dynamic initialization parameter `REMOTE_DEPENDENCIES_MODE` controls the dependency mode. The choice is either time-stamp dependency mode or RPC-signature dependency mode

Time-Stamp Dependency Mode

- Whenever a procedure is compiled, its **time stamp** is recorded in the data dictionary. The time stamp shows when the procedure was created, altered, or replaced
- A compiled procedure contains information about each remote procedure that it calls, including the schema, package name, procedure name, and time stamp of the remote procedure
- In time-stamp dependency mode, when a local stored procedure calls a remote procedure, Oracle Database compares the time stamp that the local procedure has for the remote procedure to the current time stamp of the remote procedure. If the two time stamps match, both the local and remote procedures run. Neither is recompiled
- If the two time stamps do not match, the local procedure is invalidated and an error is returned to the calling environment. All other local procedures that depend on the remote procedure with the new time stamp are also invalidated
- Time stamp comparison occurs when a statement in the body of the local procedure calls the remote procedure. Therefore, statements in the local procedure that precede the invalid call might run successfully. Statements after the invalid call do not run. The local procedure must be recompiled
- If DML statements precede the invalid call, they roll back only if they and the invalid call are in the same PL/SQL block
- The disadvantages of time-stamp dependency mode are:
    - Dependent objects across the network are often recompiled unnecessarily, degrading performance
    - If the client-side application uses PL/SQL, this mode can cause situations that prevent the application from running on the client side

RPC-Signature Dependency Mode

- Oracle Database provides **RPC signatures** to handle remote dependencies. RPC signatures do not affect local dependencies, because recompilation is always possible in the local environment

- An RPC signature is associated with each compiled stored program unit. It identifies the unit by these characteristics:

    - Name
    - Number of parameters
    - Data type class of each parameter
    - Mode of each parameter
    - Data type class of return value (for a function)

- An RPC signature changes only when at least one of the preceding characteristics changes
- > Note: An RPC signature does not include `DETERMINISTIC`, `PARALLEL_ENABLE`, or purity information. If these settings change for a function on remote system, optimizations based on them are not automatically reconsidered. Therefore, calling the remote function in a SQL statement or using it in a function-based index might cause incorrect query results
- A compiled program unit contains the RPC signature of each remote procedure that it calls (and the schema, package name, procedure name, and time stamp of the remote procedure)
- In RPC-signature dependency mode, when a local program unit calls a subprogram in a remote program unit, the database ignores time-stamp mismatches and compares the RPC signature that the local unit has for the remote subprogram to the current RPC signature of the remote subprogram. If the RPC signatures match, the call succeeds; otherwise, the database returns an error to the local unit, and the local unit is invalidated
- **Changing Names and Default Values of Parameters**
    - Changing the name or default value of a subprogram parameter does not change the RPC signature of the subprogram
    - However, if your application requires that callers get the new default value, you must recompile the called procedure
- **Changing Specification of Parameter Mode IN**
    - Because the subprogram parameter mode IN is the default, you can specify it either implicitly or explicitly
    - Changing its specification from implicit to explicit, or the reverse, does not change the RPC signature of the subprogram
- **Changing Subprogram Body**
    - Changing the body of a subprogram does not change the RPC signature of the subprogram
- **Changing Data Type Classes of Parameters**
    - Changing the data type of a parameter to another data type in the same class does not change the RPC signature, but changing the data type to a data type in another class does
- **Changing Package Types**
    - Changing the name of a package type, or the names of its internal components, does not change the RPC signature of the package

Controlling Dependency Mode

- The dynamic initialization parameter `REMOTE_DEPENDENCIES_MODE` controls the dependency mode
- If the initialization parameter file contains this specification, then only time stamps are used to resolve dependencies (if this is not explicitly overridden dynamically):

```
REMOTE_DEPENDENCIES_MODE = TIMESTAMP
```

- If the initialization parameter file contains this parameter specification, then RPC signatures are used to resolve dependencies (if this not explicitly overridden dynamically):

```
REMOTE_DEPENDENCIES_MODE = SIGNATURE
```

- You can alter the mode dynamically by using the DDL statements.

```
ALTER SESSION SET REMOTE_DEPENDENCIES_MODE = {SIGNATURE | TIMESTAMP}
ALTER SYSTEM SET REMOTE_DEPENDENCIES_MODE = {SIGNATURE | TIMESTAMP}
```

- If the `REMOTE_DEPENDENCIES_MODE` parameter is not specified, either in the init.ora parameter file or using the `ALTER SESSION` or `ALTER SYSTEM` statements, `TIMESTAMP` is the default value
- Therefore, unless you explicitly use the `REMOTE_DEPENDENCIES_MODE` parameter, or the appropriate DDL statement, your server is operating using the time-stamp dependency mode
- When you use `REMOTE_DEPENDENCIES_MODE=SIGNATURE`:
    - If you change the initial value of a parameter of a remote procedure, then the local procedure calling the remote procedure is not invalidated. If the call to the remote procedure does not supply the parameter, then the initial value is used. In this case, because invalidation and recompilation does not automatically occur, the old initial value is used. To see the new initial values, recompile the calling procedure manually
    - If you add an overloaded procedure in a package (a procedure with the same name as an existing one), then local procedures that call the remote procedure are not invalidated. If it turns out that this overloading results in a rebinding of existing calls from the local procedure under the time-stamp mode, then this rebinding does not happen under the RPC signature mode, because the local procedure does not get invalidated. You must recompile the local procedure manually to achieve the rebinding
    - If the types of parameters of an existing package procedure are changed so that the new types have the same shape as the old ones, then the local calling procedure is not invalidated or recompiled automatically. You must recompile the calling procedure manually to get the semantics of the new type
- **Dependency Resolution**
    - When `REMOTE_DEPENDENCIES_MODE = TIMESTAMP` (the default value), dependencies among program units are handled by comparing time stamps at runtime
    - If the time stamp of a called remote procedure does not match the time stamp of the called procedure, then the calling (dependent) unit is invalidated and must be recompiled. In this case, if there is no local PL/SQL compiler, then the calling application cannot proceed

    - In the time-stamp dependency mode, RPC signatures are not compared. If there is a local PL/SQL compiler, then recompilation happens automatically when the calling procedure is run
    - When `REMOTE_DEPENDENCIES_MODE = SIGNATURE`, the recorded time stamp in the calling unit is first compared to the current time stamp in the called remote unit. If they match, then the call proceeds. If the time stamps do not match, then the RPC signature of the called remote subprogram, as recorded in the calling subprogram, is compared with the current RPC signature of the called subprogram. If they do not match, then an error is returned to the calling session

- **Suggestions for Managing Dependencies**

    - Server-side PL/SQL users can set the parameter to `TIMESTAMP` (or let it default to that) to get the time-stamp dependency mode
    - Server-side PL/SQL users can use RPC-signature dependency mode if they have a distributed system and they want to avoid possible unnecessary recompilations
    - Client-side PL/SQL users must set the parameter to `SIGNATURE`. This allows:
        - Installation of applications at client sites without recompiling procedures
        - Ability to upgrade the server, without encountering time stamp mismatches
    - When using RPC signature mode on the server side, add procedures to the end of the procedure (or function) declarations in a package specification. Adding a procedure in the middle of the list of declarations can cause unnecessary invalidation and recompilation of dependent procedures

Shared SQL Dependency Management

- In addition to managing dependencies among schema objects, Oracle Database also manages dependencies of each shared SQL area in the shared pool
- If a table, view, synonym, or sequence is created, altered, or dropped, or a procedure or package specification is recompiled, all dependent shared SQL areas are invalidated
- At a subsequent execution of the cursor that corresponds to an invalidated shared SQL area, Oracle Database reparses the SQL statement to regenerate the shared SQL area