# 32
# Understanding Schema Object Dependency

If the definition of object A references object B, then A depends on B. This chapter explains dependencies among schema objects, and how Oracle Database automatically tracks and manages these dependencies. Because of this automatic dependency management, A never uses an obsolete version of B, and you almost never have to explicitly recompile A after you change B.

**Topics:**



## 32.1 Overview of Schema Object Dependency

Some types of schema objects can reference other objects in their definitions. For example, a view is defined by a query that references tables or other 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).

### 32.1.1 Example: Displaying Dependent and Referenced Object Types

Example 32-1 shows how to display the dependent and referenced object types in your database (if you are logged in as DBA).

**Example 32-1 Displaying Dependent and Referenced Object Types**

Display dependent object types:

```
SELECT DISTINCT TYPE
FROM DBA_DEPENDENCIES
ORDER BY TYPE;
```

Result:

```
TYPE
------------------
DIMENSION
```

```
EVALUATION CONTXT
FUNCTION
INDEX
INDEXTYPE
JAVA CLASS
JAVA DATA
MATERIALIZED VIEW
OPERATOR
PACKAGE
PACKAGE BODY
PROCEDURE
RULE
RULE SET
SYNONYM
TABLE
TRIGGER
TYPE
TYPE BODY
UNDEFINED
VIEW
XML SCHEMA

22 rows selected.
```

Display referenced object types:

```
SELECT DISTINCT REFERENCED_TYPE
FROM DBA_DEPENDENCIES
ORDER BY REFERENCED_TYPE;
```

Result:

```
REFERENCED_TYPE
------------------
EVALUATION CONTXT
FUNCTION
INDEX
INDEXTYPE
JAVA CLASS
LIBRARY
OPERATOR
PACKAGE
PROCEDURE
SEQUENCE
SYNONYM
TABLE
TYPE
VIEW
XML SCHEMA

14 rows selected.
```

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.

## 32.1.2 Example: Schema Object Change that Invalidates Some Dependents

As an example of a schema object change that invalidates some dependents but not others, consider the two views in the following example, which are based on the `HR.EMPLOYEES` table.

Example 32-2 creates two views from the EMPLOYEES table: SIXFIGURES, which selects all columns in the table, and COMMISSIONED, which does not include the EMAIL column. As the example shows, changing the EMAIL column invalidates SIXFIGURES, but not COMMISSIONED.

**Example 32-2 Schema Object Change That Invalidates Some Dependents**

```
CREATE OR REPLACE VIEW sixfigures AS
SELECT * FROM employees
WHERE salary >= 100000;

CREATE OR REPLACE VIEW commissioned AS
SELECT first_name, last_name, commission_pct
FROM employees
WHERE commission_pct > 0.00;
```

SQL*Plus formatting command:

```
COLUMN object_name FORMAT A16
```

Query:

```
SELECT object_name, status
FROM user_objects
WHERE object_type = 'VIEW'
ORDER BY object_name;
```

Result:

```
OBJECT_NAME      STATUS
---------------- -------
COMMISSIONED     VALID
EMP_DETAILS_VIEW VALID
SIXFIGURES       INVALID

3 rows selected.
```

Lengthen EMAIL column of EMPLOYEES table:

```
ALTER TABLE employees MODIFY email VARCHAR2(100);
```

Query:

```
SELECT object_name, status
FROM user_objects
WHERE object_type = 'VIEW'
ORDER BY object_name;
```

Result:

```
OBJECT_NAME      STATUS
---------------- -------
COMMISSIONED     VALID
EMP_DETAILS_VIEW INVALID
SIXFIGURES       VALID
```

## 32.1.3 Example: View That Depends on Multiple Objects

A view depends on every object referenced in its query. The view in Example 32-3 depends on the tables employees and departments.

**Example 32-3 View that Depends on Multiple Objects**

```
CREATE OR REPLACE VIEW v AS
  SELECT last_name, first_name, department_name
  FROM employees e, departments d
  WHERE e.department_id = d.department_id
  ORDER BY last_name;
```

Note the following:

- `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'
  ```

# 32.2 Querying Object Dependencies

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

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`.

> **See Also:**
>
> *Oracle Database Reference* for more information about the `DEPTREE`, `IDEPTREE`, and `utldtree.sql` script

# 32.3 Object Status

Every database object has a status value described in Table 32-1.

**Table 32-1 Database Object Status**

| Status | Meaning |
|---|---|
| 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.) |

> **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`.

## 32.4 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.

The operations in the left column of Table 32-2 cause fine-grained invalidation, except in the cases in the right column. The cases in the right column, and all operations not listed in Table 32-2, cause coarse-grained invalidation.

**Table 32-2 Operations that Cause Fine-Grained Invalidation**

| Operation | Exceptions |
|---|---|
| `ALTER TABLE table ADD column` | • Dependent object (except a view) uses `SELECT *` on *table*.<br>• Dependent object uses *table*`%rowtype`.<br>• Dependent object performs `INSERT` on *table* without specifying column list.<br>• Dependent object references *table* in query that contains a join.<br>• Dependent object references *table* in query that references a PL/SQL variable. |
| `ALTER TABLE table {MODIFY\|RENAME\|DROP\|SET UNUSED} column`<br>`ALTER TABLE table DROP CONSTRAINT not_null_constraint` | • Dependent object directly references *column*.<br>• Dependent object uses `SELECT *` on *table*.<br>• Dependent object uses *table*`%ROWTYPE`.<br>• Dependent object performs `INSERT` on *table* without specifying column list.<br>• Dependent object is a trigger that depends on an entire row (that is, it does not specify a column in its definition).<br>• Dependent object is a trigger that depends on a column to the right of the dropped column. |
| `CREATE OR REPLACE VIEW view`<br>Online Table Redefinition (`DBMS_REDEFINITION`) | Column lists of new and old definitions differ, and at least one of these is true:<br>• Dependent object references column that is modified or dropped in new view or table definition.<br>• Dependent object uses *view*`%rowtype` or *table*`%rowtype`.<br>• Dependent object performs `INSERT` on view or table without specifying column list.<br>• New view definition introduces new columns, and dependent object references view or table in query that contains a join.<br>• New view definition introduces new columns, and dependent object references view or table in query that references a PL/SQL variable.<br>• Dependent object references view or table in `RELIES ON` clause. |
| `CREATE OR REPLACE SYNONYM synonym` | • New and old *synonym* targets differ, and one is not a table.<br>• Both old and new *synonym* targets are tables, and the tables have different column lists or different privilege grants.<br>• Both old and new *synonym* targets are tables, and dependent object is a view that references a column that participates in a unique index on the old target but not in a unique index on the new target. |

**Table 32-2 (Cont.) Operations that Cause Fine-Grained Invalidation**

| Operation | Exceptions |
|---|---|
| `DROP INDEX` | • The index is a function-based index and the dependent object is a trigger that depends either on an entire row or on a column that was added to *`table`* after a function-based index was created.<br>• The index is a unique index, the dependent object is a view, and the view references a column participating in the unique index. |
| `CREATE OR REPLACE {PROCEDURE\|FUNCTION}` | Call signature changes. Call signature is the parameter list (order, names, and types of parameters), return type, `ACCESSIBLE BY` clause ("white list"), purity[1], determinism, parallelism, pipelining, and (if the procedure or function is implemented in C or Java) implementation properties. |
| `CREATE OR REPLACE PACKAGE` | • `ACCESSIBLE BY` clause ("white list") changes.<br>• Dependent object references a dropped or renamed package item.<br>• Dependent object references a package procedure or function whose call signature or entry-point number[2], changed.<br>If referenced procedure or function has multiple overload candidates, dependent object is invalidated if any overload candidate's call signature or entry point number changed, or if a candidate was added or dropped.<br>• Dependent object references a package cursor whose call signature, rowtype, or entry point number changed.<br>• Dependent object references a package type or subtype whose definition changed.<br>• Dependent object references a package variable or constant whose name, data type, initial value, or offset number changed.<br>• Package purity[1] changed. |

1 **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.

2 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.

> **Note:**
>
> A dependent object that is invalidated by an operation in Table 32-2 appears in the static data dictionary views `*_OBJECTS` and `*_OBJECTS_AE` only after an attempt to reference it (either during compilation or execution) or after invoking one of these subprograms:
>
> - `DBMS_UTILITY.COMPILE_SCHEMA`
> - Any `UTL_RECOMP` subprogram

**Topics:**

- Session State and Referenced Packages
- Security Authorization

> **See Also:**
>
> - *Oracle Database PL/SQL Language Reference* for information about PL/SQL compilation parameter settings
> - *Oracle Database PL/SQL Packages and Types Reference* for more information about `DBMS_UTILITY.COMPILE_SCHEMA`
> - *Oracle Database PL/SQL Packages and Types Reference* for more information about `UTL_RECOMP` subprogram

## 32.4.1 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.

> **See Also:**
>
> - *Oracle Database PL/SQL Language Reference* for information about package instantiation
> - *Oracle Database PL/SQL Language Reference* for information about package state

## 32.4.2 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.

# 32.5 Guidelines for Reducing Invalidation

To reduce invalidation of dependent objects, follow these guidelines:

- Add Items to End of Package
- Reference Each Table Through a View

## 32.5.1 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.

For example, consider this package:

```
CREATE OR REPLACE PACKAGE pkg1 AUTHID DEFINER IS
  FUNCTION get_var RETURN VARCHAR2;
END;
/
```

Adding an item to the end of `pkg1`, as follows, does not invalidate dependents that reference the `get_var` function:

```
CREATE OR REPLACE PACKAGE pkg1 AUTHID DEFINER IS
  FUNCTION get_var RETURN VARCHAR2;
  PROCEDURE set_var (v VARCHAR2);
END;
/
```

Inserting an item between the `get_var` function and the `set_var` procedure, as follows, invalidates dependents that reference the `set_var` function:

```
CREATE OR REPLACE PACKAGE pkg1 AUTHID DEFINER IS
  FUNCTION get_var RETURN VARCHAR2;
  PROCEDURE assert_var (v VARCHAR2);
  PROCEDURE set_var (v VARCHAR2);
END;
/
```

## 32.5.2 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`.

# 32.6 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. For definitions of these terms, see Table 32-1.

**Topics:**

- Revalidation of Objects that Compiled with Errors
- Revalidation of Unauthorized Objects
- Revalidation of Invalid SQL Objects
- Revalidation of Invalid PL/SQL Objects

## 32.6.1 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.

## 32.6.2 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.

## 32.6.3 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.

## 32.6.4 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.

# 32.7 Name Resolution in Schema Scope

Object names referenced in SQL statements have one or more pieces. Pieces are separated by periods—for example, `hr.employees.department_id` has three pieces.

Oracle Database uses the following procedure to try to resolve an object name.

> **Note:**
>
> For the procedure to succeed, all pieces of the object name must be visible in the current edition.

1. Try to qualify the first piece of the object name.

If the object name has only one piece, then that piece is the first piece. Otherwise, the first piece is the piece to the left of the leftmost period; for example, in `hr.employees.department_id`, the first piece is `hr`.

The procedure for trying to qualify the first piece is:

a. If the object name is a table name that appears in the `FROM` clause of a `SELECT` statement, and the object name has multiple pieces, go to step d. Otherwise, go to step b.

   Search the current schema for an object whose name matches the first piece.

   If found, go to step 2. Otherwise, go to step c.

b. Search for a public synonym that matches the first piece.

   If found, go to step 2. Otherwise, go to step d.

c. Search for a schema whose name matches the first piece.

   If found, and if the object name has a second piece, go to step e. Otherwise, return an error—the object name cannot be qualified.

d. Search the schema found at step d for a table or SQL function whose name matches the second piece of the object name.

   If found, go to step 2. Otherwise, return an error—the object name cannot be qualified.

> **Note:**
>
> A SQL function found at this step has been redefined by the schema found at step d.

2. A schema object has been qualified. Any remaining pieces of the object name must match a valid part of this schema object.

   For example, if the object name is `hr.employees.department_id`, `hr` is qualified as a schema. If `employees` is qualified as a table, `department_id` must correspond to a column of that table. If `employees` is qualified as a package, `department_id` must correspond to a public constant, variable, procedure, or function of that package.

Because of how Oracle Database resolves references, an object can depend on the nonexistence of other objects. This situation occurs when the dependent object uses a reference that would be interpreted differently if another object were present.

> **See Also:**
>
> - *Oracle Database PL/SQL Language Reference* for information about how name resolution differs in SQL and PL/SQL
> - *Oracle Database Administrator's Guide* for information about name resolution in a distributed database system
> - Name Resolution for Editioned and Noneditioned Objects

# 32.8 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.

# 32.9 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.

**Topics:**



## 32.9.1 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.

> **See Also:**
>
> - *Oracle Database PL/SQL Language Reference*
> - Time-Stamp Dependency Mode
> - RPC-Signature Dependency Mode

## 32.9.2 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.

However, the local procedure may be invalidated when a new procedure that references the same remote table is created. The reason being the local procedure is actually dependent on the stub object representing the remote table and it is invalidated when the stub object's timestamp is modified. The stub object's timestamp refresh is strictly limited to scenarios such as create, alter, drop and reload dependencies. Therefore, though execution of the procedure does not refresh the timestamp of the stub object (in order to avoid huge overhead causing performance degrade), it happens during the creation of another local procedure referring the remote table or compilation of the current procedure itself.

### 32.9.3 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.

> **See Also:**
>
> Manuals for your application development tools and your operating system for more information about managing the remote dependencies within database applications

## 32.10 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.

**Topics:**



### 32.10.1 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. For example, the `UPDATE` statement rolls back in this code:

```
BEGIN

  UPDATE table SET ...
  invalid_proc;
  COMMIT;

END;
```

But the `UPDATE` statement does not roll back in this code:

```
UPDATE table SET ...
EXECUTE invalid_proc;
COMMIT;
```

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.

  An example of such an application is Oracle Forms. During installation, you must recompile the client-side PL/SQL procedures that Oracle Forms uses at the client site. Also, if a client-side procedure depends on a server procedure, and if the server procedure changes or is automatically recompiled, you must recompile the client-side PL/SQL procedure. However, no PL/SQL compiler is available on the client. Therefore, the developer of the client application must distribute new versions of the application to all customers.

## 32.10.2 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.

For example, suppose that this procedure, `get_emp_name`, is stored on a server in Boston (`BOSTON_SERVER`):

```
CREATE OR REPLACE PROCEDURE get_emp_name (
  emp_number  IN  NUMBER,
  hiredate    OUT VARCHAR2,
  emp_name    OUT VARCHAR2) AUTHID DEFINER AS
BEGIN
  SELECT last_name, TO_CHAR(hire_date, 'DD-MON-YY')
  INTO emp_name, hiredate
  FROM employees
  WHERE employee_id = emp_number;
END;
/
```

When `get_emp_name` is compiled on `BOSTON_SERVER`, Oracle Database records both its RPC signature and its time stamp.

Suppose that this PL/SQL procedure, `print_name`, which calls `get_emp_name`, is on a server in California:

```
CREATE OR REPLACE PROCEDURE print_ename (emp_number IN NUMBER) AUTHID DEFINER AS
  hiredate  VARCHAR2(12);
  ename     VARCHAR2(10);
BEGIN
  get_emp_name@BOSTON_SERVER(emp_number, hiredate, ename);
  dbms_output.put_line(ename);
  dbms_output.put_line(hiredate);
END;
/
```

When `print_name` is compiled on the California server, the database connects to the Boston server, sends the RPC signature of `get_emp_name` to the California server, and records the RPC signature of `get_emp_name` in the compiled state of `print_ename`.

At runtime, when `print_name` calls `get_emp_name`, the database sends the RPC signature of `get_emp_name` that was recorded in the compiled state of `print_ename` to the Boston server. If the recorded RPC signature matches the current RPC signature of `get_emp_name` on the Boston server, the call succeeds; otherwise, the database returns an error to `print_name`, which is invalidated.

**Topics:**

- Changing Names and Default Values of Parameters
- Changing Specification of Parameter Mode IN
- Changing Subprogram Body
- Changing Data Type Classes of Parameters
- Changing Package Types

## 32.10.2.1 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. For example, procedure `P1` has the same RPC signature in these two examples:

```
PROCEDURE P1 (Param1 IN NUMBER := 100);
PROCEDURE P1 (Param2 IN NUMBER := 200);
```

However, if your application requires that callers get the new default value, you must recompile the called procedure.

## 32.10.2.2 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. For example, procedure `P1` has the same RPC signature in these two examples:

```
PROCEDURE P1 (Param1 NUMBER);    -- implicit specification
PROCEDURE P1 (Param1 IN NUMBER);  -- explicit specification
```

## 32.10.2.3 Changing Subprogram Body

Changing the body of a subprogram does not change the RPC signature of the subprogram.

Example 32-4 changes only the body of the procedure `get_hire_date`; therefore, it does not change the RPC signature of `get_hire_date`.

**Example 32-4 Changing Body of Procedure get_hire_date**

```
CREATE OR REPLACE PROCEDURE get_hire_date (
  emp_number  IN  NUMBER,
  hiredate    OUT VARCHAR2,
  emp_name    OUT VARCHAR2) AUTHID DEFINER AS
BEGIN
  SELECT last_name, TO_CHAR(hire_date, 'DD-MON-YY')
  INTO emp_name, hiredate
  FROM employees
  WHERE employee_id = emp_number;
END;
/
```

```
CREATE OR REPLACE PROCEDURE get_hire_date (
  emp_number  IN  NUMBER,
  hiredate    OUT VARCHAR2,
  emp_name    OUT VARCHAR2) AUTHID DEFINER AS
BEGIN
  -- Change date format model
  SELECT last_name, TO_CHAR(hire_date, 'DD/MON/YYYY')
  INTO emp_name, hiredate
  FROM employees
  WHERE employee_id = emp_number;
END;
/
```

## 32.10.2.4 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.

Table 32-3 lists the data type classes and the data types that comprise them. Data types not listed in Table 32-3, such as `NCHAR`, do not belong to a data type class. Changing their type always changes the RPC signature.

**Table 32-3 Data Type Classes**

| Data Type Class | Data Types in Class |
|---|---|
| Character | CHAR<br>CHARACTER |
| VARCHAR | VARCHAR<br>VARCHAR2<br>STRING<br>LONG<br>ROWID |
| Raw | RAW<br>LONG RAW |
| Integer | BINARY_INTEGER<br>PLS_INTEGER<br>SIMPLE_INTEGER<br>BOOLEAN<br>NATURAL<br>NATURALN<br>POSITIVE<br>POSITIVEN |

**Table 32-3 (Cont.) Data Type Classes**

| Data Type Class | Data Types in Class |
|---|---|
| Number | NUMBER<br>INT<br>INTEGER<br>SMALLINT<br>DEC<br>DECIMAL<br>REAL<br>FLOAT<br>NUMERIC<br>DOUBLE PRECISION |
| Datetime | DATE<br>TIMESTAMP<br>TIMESTAMP WITH TIME ZONE<br>TIMESTAMP WITH LOCAL TIME ZONE<br>INTERVAL YEAR TO MONTH<br>INTERVAL DAY TO SECOND |

Example 32-5 changes the data type of the parameter `hiredate` from `VARCHAR2` to `DATE`. `VARCHAR2` and `DATE` are not in the same data type class, so the RPC signature of the procedure `get_hire_date` changes.

**Example 32-5 Changing Data Type Class of get_hire_date Parameter**

```
CREATE OR REPLACE PROCEDURE get_hire_date (
  emp_number  IN  NUMBER,
  hiredate    OUT DATE,
  emp_name    OUT VARCHAR2) AS
BEGIN
  SELECT last_name, TO_CHAR(hire_date, 'DD/MON/YYYY')
  INTO emp_name, hiredate
  FROM employees
  WHERE employee_id = emp_number;
END;
/
```

## 32.10.2.5 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.

Example 32-6 defines a record type, `emp_data_type`, inside the package `emp_package`. Next, it changes the names of the record fields, but not their types. Finally, it changes the name of the type, but not its characteristics. The RPC signature of the package does not change.

**Example 32-6 Changing Names of Fields in Package Record Type**

```
CREATE OR REPLACE PACKAGE emp_package AUTHID DEFINER AS
  TYPE emp_data_type IS RECORD (
    emp_number  NUMBER,
    hiredate    VARCHAR2(12),
    emp_name    VARCHAR2(10)
  );
  PROCEDURE get_emp_data (
```

```
    emp_data  IN OUT emp_data_type
  );
END;
/

CREATE OR REPLACE PACKAGE emp_package AUTHID DEFINER AS
  TYPE emp_data_type IS RECORD (
    emp_num   NUMBER,
    hire_dat  VARCHAR2(12),
    empname   VARCHAR2(10)
  );
  PROCEDURE get_emp_data (
    emp_data  IN OUT emp_data_type
  );
END;
/

CREATE OR REPLACE PACKAGE emp_package AUTHID DEFINER AS
  TYPE emp_data_record_type IS RECORD (
    emp_num   NUMBER,
    hire_dat  VARCHAR2(12),
    empname   VARCHAR2(10)
  );
  PROCEDURE get_emp_data (
    emp_data  IN OUT emp_data_record_type
  );
END;
/
```

## 32.10.3 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. For example, this example alters the dependency mode for the current session:

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

This example alters the dependency mode systemwide after startup:

```
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.

**Topics:**



## 32.10.3.1 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 (using the criteria described in Changing Data Type Classes of Parameters), then an error is returned to the calling session.

## 32.10.3.2 Suggestions for Managing Dependencies

Follow these guidelines for setting the `REMOTE_DEPENDENCIES_MODE` parameter:

- 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.

# 32.11 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.