ALTER TYPE

Purpose

Use the `ALTER TYPE` statement to add or drop member attributes or methods. You can change the existing properties (`FINAL` or `INSTANTIABLE`) of an object type, and you can modify the scalar attributes of the type.

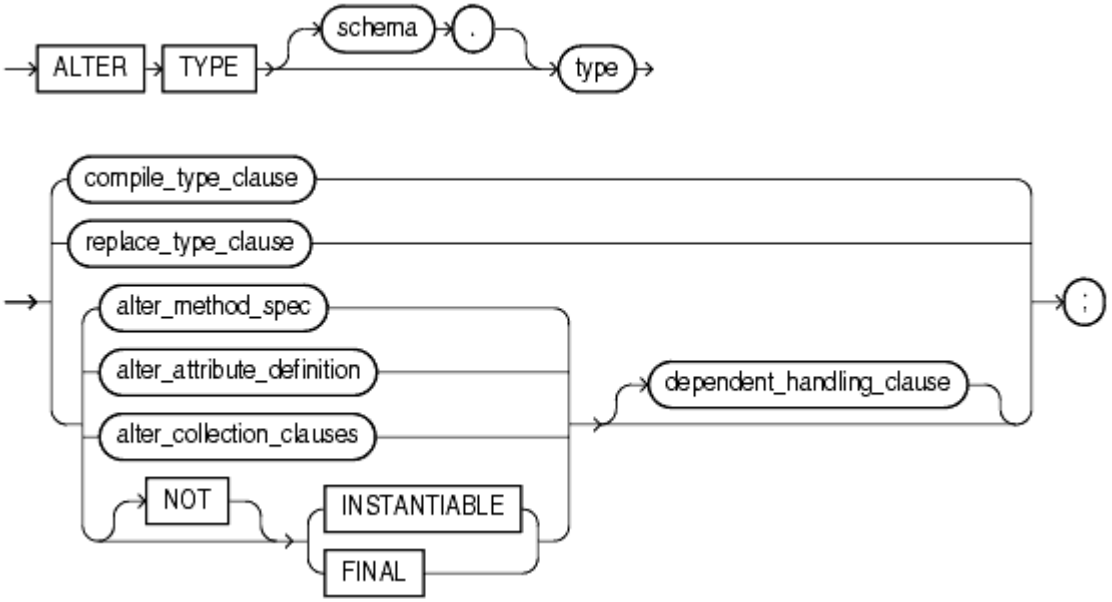
You can also use this statement to recompile the specification or body of the type or to change the specification of an object type by adding new object member subprogram specifications.

Prerequisites

The object type must be in your own schema and you must have `CREATE TYPE` or `CREATE ANY TYPE` system privilege, or you must have `ALTER ANY TYPE` system privileges.

Syntax

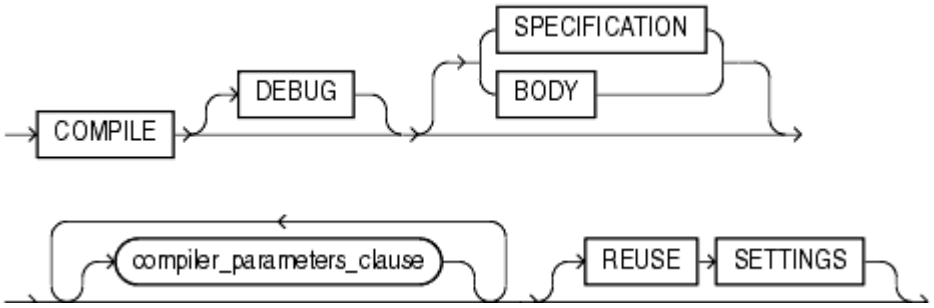
alter_type::=



Description of the illustration alter_type.gif

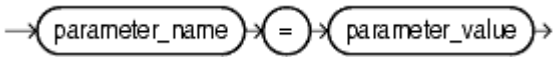
(*compile_type_clause ::=*, *replace_type_clause ::=*, *alter_method_spec ::=*, *alter_attribute_definition::=*, *alter_collection_clauses ::=*, *dependent_handling_clause ::=*)

compile_type_clause **::=**



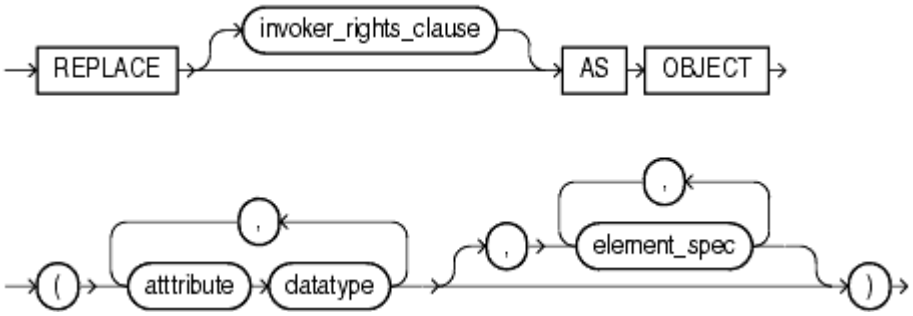
Description of the illustration compile_type_clause.gif

compiler_parameters_clause::=



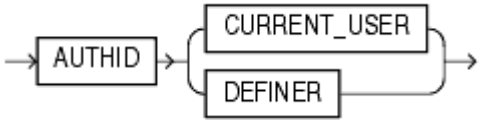
Description of the illustration compiler_parameters_clause.gif

replace_type_clause **::=**



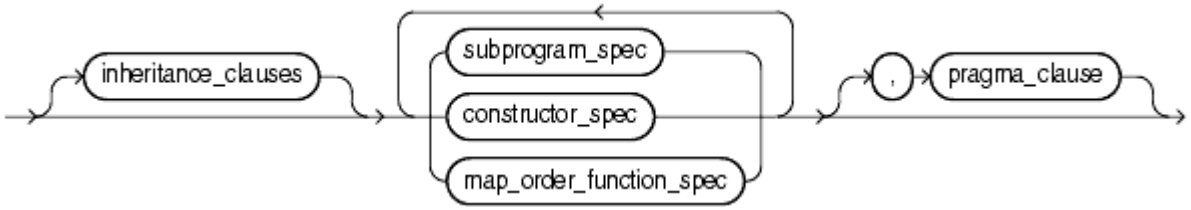
Description of the illustration replace_type_clause.gif

invoker_rights_clause::=



Description of the illustration invoker_rights_clause.gif

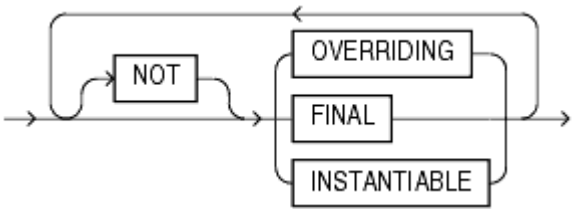
element_spec **::=**



Description of the illustration element_spec.gif

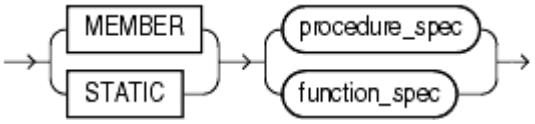
(*inheritance_clauses::=*, *subprogram_spec::=*, *constructor_spec::=*, *map_order_function_spec::=*, *pragma_clause ::=*)

inheritance_clauses::=



Description of the illustration inheritance_clauses.gif

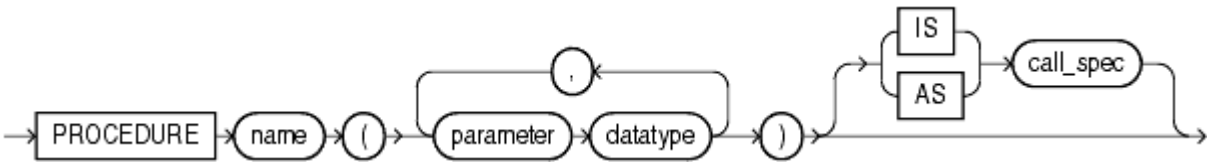
subprogram_spec::=



Description of the illustration subprogram_spec.gif

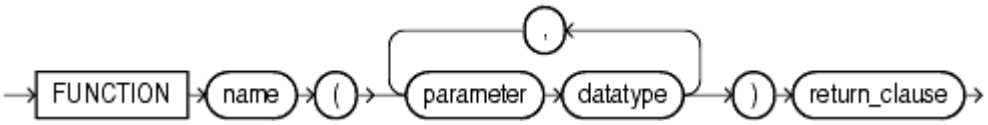
(*procedure_spec::=*, *function_spec::=*)

procedure_spec::=



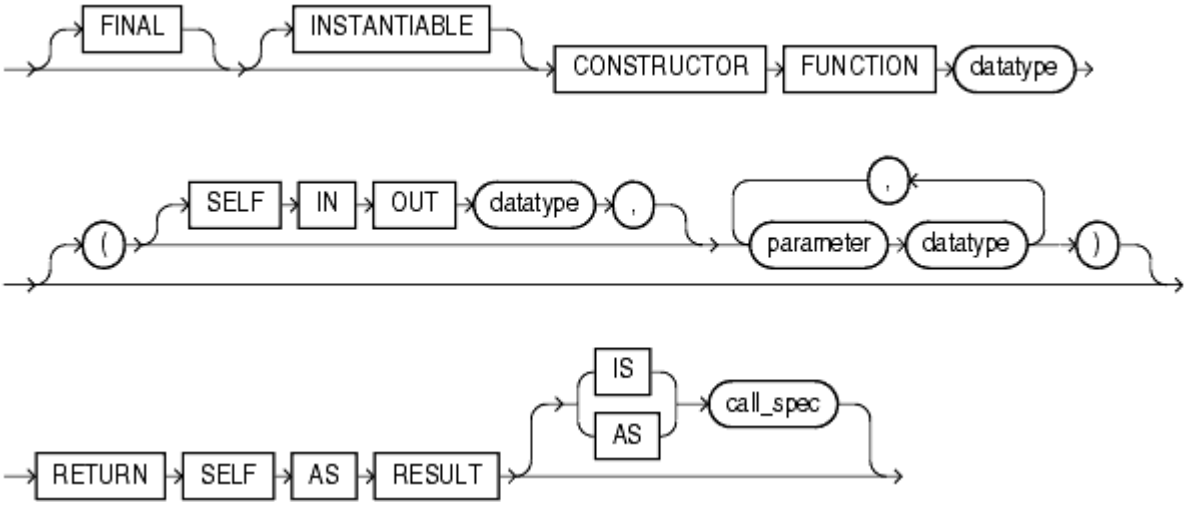
Description of the illustration procedure_spec.gif

function_spec::=



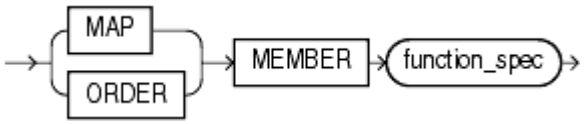
Description of the illustration function_spec.gif

constructor_spec::=



Description of the illustration constructor_spec.gif

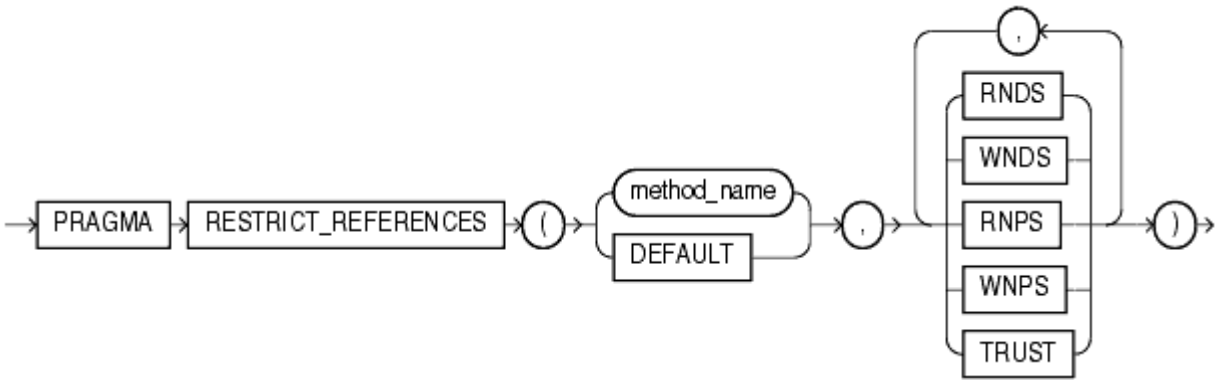
map_order_function_spec::=



Description of the illustration map_order_function_spec.gif

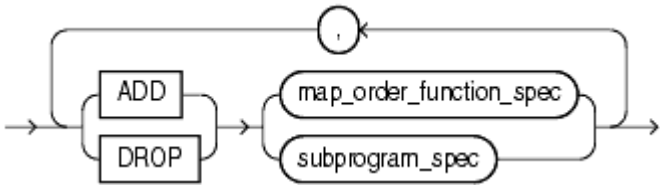
(*function_spec::=*)

pragma_clause **::=**



Description of the illustration pragma_clause.gif

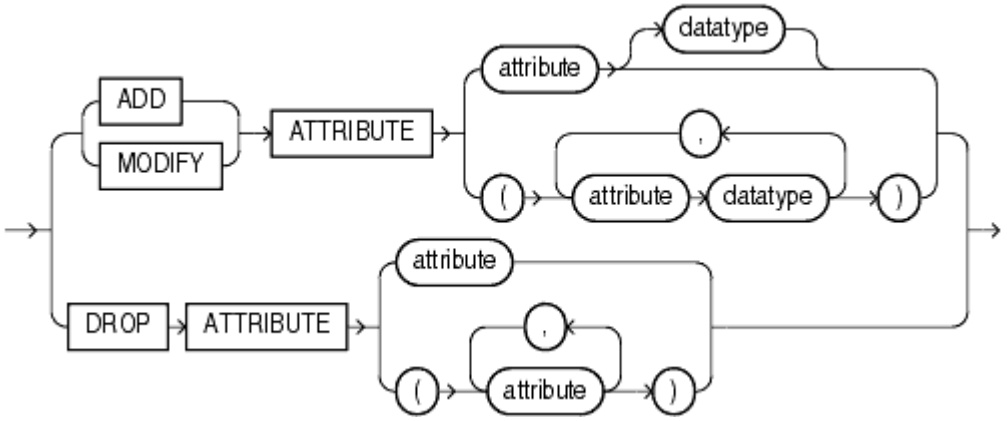
alter_method_spec **::=**



Description of the illustration alter_method_spec.gif

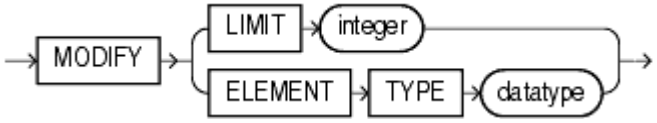
(*map_order_function_spec::=*, *subprogram_spec::=*)

*alter_attribute_definition***::=**



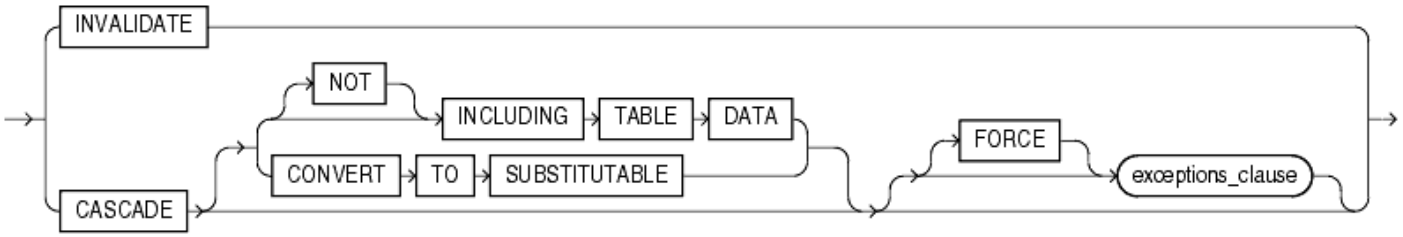
Description of the illustration alter_attribute_definition.gif

alter_collection_clauses **::=**



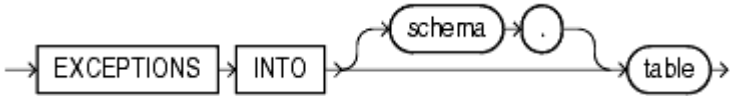
Description of the illustration alter_collection_clauses.gif

dependent_handling_clause **::=**



Description of the illustration dependent_handling_clause.gif

exceptions_clause::=



Description of the illustration exceptions_clause.gif

Semantics

schema

Specify the schema that contains the type. If you omit *schema*, then Oracle Database assumes the type is in your current schema.

type

Specify the name of an object type, a nested table type, or a varray type.

compile_type_clause

Specify `COMPILE` to compile the object type specification and body. This is the default if neither `SPECIFICATION` nor `BODY` is specified.

During recompilation, Oracle Database drops all persistent compiler switch settings, retrieves them again from the session, and stores them at the end of compilation. To avoid this process, specify the `REUSE SETTINGS` clause.

If recompiling the type results in compilation errors, then the database returns an error and the type remains invalid. You can see the associated compiler error messages with the SQL*Plus command `SHOW ERRORS`.

DEBUG

Specify `DEBUG` to instruct the PL/SQL compiler to generate and store the code for use by the PL/SQL debugger. Specifying this clause has the same effect as specifying `PLSQL_DEBUG = TRUE` in the *compiler_parameters_clause*.

SPECIFICATION

Specify SPECIFICATION to compile only the object type specification.

BODY

Specify BODY to compile only the object type body.

compiler_parameters_clause

Use this clause to specify a value for one of the PL/SQL compiler parameters. The parameters you can specify in this clause are PLSQL_OPTIMIZE_LEVEL, PLSQL_CODE_TYPE, PLSQL_DEBUG, PLSQL_WARNINGS, and NLS_LENGTH_SEMANTICS.

You can specify each parameter only once in each statement. Each setting is valid only for the current library unit being compiled and does not affect other compilations in this session or system. To affect the entire session or system, you must set a value for the parameter using the ALTER SESSION or ALTER SYSTEM statement.

If you omit any parameter from this clause and you specify REUSE SETTINGS, then if a value was specified for the parameter in an earlier compilation of this library unit, Oracle Database uses that earlier value. If you omit any parameter and either you do not specify REUSE SETTINGS or no value has been specified for the parameter in an earlier compilation, then the database obtains the value for that parameter from the session environment.

Restriction on the *compiler_parameters_clause*

You cannot set a value for the PLSQL_DEBUG parameter if you also specify DEBUG, because both clauses set the PLSQL_DEBUG parameter, and you can specify a value for each parameter only once.

REUSE SETTINGS

Specify REUSE SETTINGS to prevent Oracle from dropping and reacquiring compiler switch settings. With this clause, Oracle preserves the existing settings and uses them for the recompilation of any parameters for which values are not specified elsewhere in this statement.

For backward compatibility, Oracle Database sets the persistently stored value of the PLSQL_COMPILER_FLAGS initialization parameter to reflect the values of the PLSQL_CODE_TYPE and PLSQL_DEBUG parameters that result from this statement.

replace_type_clause

The REPLACE clause lets you add new member subprogram specifications. This clause is valid only for object types, not for nested tables or varrays.

attribute

Specify an object attribute name. Attributes are data items with a name and a type specifier that form the structure of the object.

element_spec

Specify the elements of the redefined object.

inheritance_clauses

The inheritance_clauses have the same semantics in `CREATE TYPE` and `ALTER TYPE` statements. Please refer to *inheritance_clauses* in the documentation on `CREATE TYPE`.

subprogram_spec

The `MEMBER` and `STATIC` clauses let you specify for the object type a function or procedure subprogram which is referenced as an attribute.

You must specify a corresponding method body in the object type body for each procedure or function specification.

See Also:

CREATE TYPE for a description of the difference between member and static methods, and for examples

PL/SQL User's Guide and Reference for information about overloading subprogram names within a package

CREATE TYPE BODY

procedure_spec

Enter the specification of a procedure subprogram.

function_spec

Enter the specification of a function subprogram.

pragma_clause

The *`pragma_clause`* is a compiler directive that denies member functions read/write access to database tables, packaged variables, or both, and thereby helps to avoid side effects.

Oracle recommends that you avoid using this clause unless you must do so for backward compatibility of your applications. This clause has been deprecated. Oracle Database now runs purity checks at run time. If you must use this clause for backward compatibility of your applications, you can find its description in *pragma_clause* (under `CREATE TYPE`).

Restriction on Pragmas

The *pragma_clause* is not valid when dropping a method.

map_order_function_spec

You can declare either one MAP method or one ORDER method, regardless how many MEMBER or STATIC methods you declare. However, a subtype can override a MAP method if the supertype defines a NOT FINAL MAP method. If you declare either method, then you can compare object instances in SQL.

If you do not declare either method, then you can compare object instances only for equality or inequality. Instances of the same type definition are equal only if each pair of their corresponding attributes is equal. No comparison method needs to be specified to determine the equality of two object types.

For MAP, specify a member function (MAP method) that returns the relative position of a given instance in the ordering of all instances of the object. A map method is called implicitly and induces an ordering of object instances by mapping them to values of a predefined scalar type. Oracle Database uses the ordering for comparison conditions and ORDER BY clauses.

If *type* will be referenced in queries involving sorts (through ORDER BY, GROUP BY, DISTINCT, or UNION clauses) or joins, and you want those queries to be parallelized, then you must specify a MAP member function.

If the argument to the MAP method is null, then the MAP method returns null and the method is not invoked.

An object specification can contain only one MAP method, which must be a function. The result type must be a predefined SQL scalar type, and the MAP function can have no arguments other than the implicit SELF argument.

A subtype cannot define a new MAP method. However, it can override an inherited MAP method.

For ORDER, specify a member function (ORDER method) that takes an instance of an object as an explicit argument and the implicit SELF argument and returns either a negative, zero, or positive integer. The negative, zero, or positive value indicates that the implicit SELF argument is less than, equal to, or greater than the explicit argument.

If either argument to the ORDER method is null, then the ORDER method returns null and the method is not invoked.

When instances of the same object type definition are compared in an ORDER BY clause, the ORDER method function is invoked.

An object specification can contain only one ORDER method, which must be a function having the

return type NUMBER.

A subtype cannot define an ORDER method, nor can it override an inherited ORDER method.

invoker_rights_clause

The *invoker_rights_clause* lets you specify whether the member functions and procedures of the object type execute with the privileges and in the schema of the user who owns the object type or with the privileges and in the schema of CURRENT_USER. This specification applies to the corresponding type body as well.

This clause also determines how Oracle Database resolves external names in queries, DML operations, and dynamic SQL statements in the member functions and procedures of the type.

Restriction on Invoker Rights

You can specify this clause only for an object type, not for a nested table or varray.

AUTHID CURRENT_USER Clause

Specify CURRENT_USER if you want the member functions and procedures of the object type to execute with the privileges of CURRENT_USER. This clause creates an **invoker-rights type**.

You must specify this clause to maintain invoker-rights status for the type if you created it with this status. Otherwise the status will revert to definer rights.

This clause also specifies that external names in queries, DML operations, and dynamic SQL statements resolve in the schema of CURRENT_USER. External names in all other statements resolve in the schema in which the type resides.

AUTHID DEFINER Clause

Specify DEFINER if you want the member functions and procedures of the object type to execute with the privileges of the owner of the schema in which the functions and procedures reside, and that external names resolve in the schema where the member functions and procedures reside. This is the default.

See Also:

Oracle Database Concepts and *Oracle Database Application Developer's Guide - Fundamentals* for information on how CURRENT_USER is determined

PL/SQL User's Guide and Reference

alter_method_spec

The *alter_method_spec* lets you add a method to or drop a method from *type*. Oracle Database disables any function-based indexes that depend on the type.

In one ALTER TYPE statement you can add or drop multiple methods, but you can reference each method only once.

ADD

When you add a method, its name must not conflict with any existing attributes in its type hierarchy.

DROP

When you drop a method, Oracle Database removes the method from the target type.

Restriction on Dropping Methods

You cannot drop from a subtype a method inherited from its supertype. Instead you must drop the method from the supertype.

subprogram_spec

The MEMBER and STATIC clauses let you add a procedure subprogram to or drop it from the object type.

Restriction on Subprograms

You cannot define a STATIC method on a subtype that redefines a MEMBER method in its supertype, or vice versa. Please refer to the description of the *subprogram_spec* in CREATE TYPE for more information.

map_order_function_spec

If you declare either a MAP or ORDER method, then you can compare object instances in SQL.

Restriction on MAP and ORDER Methods

You cannot add an ORDER method to a subtype. Please refer to the description of *constructor_spec* in CREATE TYPE for more information.

alter_attribute_definition

The *alter_attribute_definition* clause lets you add, drop, or modify an attribute of an object type. In one ALTER TYPE statement, you can add, drop, or modify multiple member attributes or methods, but you can reference each attribute or method only once.

ADD ATTRIBUTE

The name of the new attribute must not conflict with existing attributes or methods in the type hierarchy. Oracle Database adds the new attribute to the end of the locally defined attribute list.

If you add the attribute to a supertype, then it is inherited by all of its subtypes. In subtypes, inherited attributes always precede declared attributes. Therefore, you may need to update the mappings of the implicitly altered subtypes after adding an attribute to a supertype.

DROP ATTRIBUTE

When you drop an attribute from a type, Oracle Database drops the column corresponding to the dropped attribute as well as any indexes, statistics, and constraints referencing the dropped attribute.

You need not specify the datatype of the attribute you are dropping.

Restrictions on Dropping Type Attributes

- You cannot drop an attribute inherited from a supertype. Instead you must drop the attribute from the supertype.
- You cannot drop an attribute that is part of a partitioning, subpartitioning, or cluster key.
- You cannot drop an attribute of a primary-key-based object identifier of an object table or a primary key of an index-organized table.
- You cannot drop all of the attributes of a root type. Instead you must drop the type. However, you can drop all of the locally declared attributes of a subtype.

MODIFY ATTRIBUTE

This clause lets you modify the datatype of an existing scalar attribute. For example, you can increase the length of a `VARCHAR2` or `RAW` attribute, or you can increase the precision or scale of a numeric attribute.

Restriction on Modifying Attributes

You cannot expand the size of an attribute referenced in a function-based index, domain index, or cluster key.

[NOT] FINAL

Use this clause to indicate whether any further subtypes can be created for this type:

Specify `FINAL` if no further subtypes can be created for this type.

Specify `NOT FINAL` if further subtypes can be created under this type.

If you change the property between `FINAL` and `NOT FINAL`, then you must specify the `CASCADE` clause of the *dependent_handling_clause* to convert data in dependent columns and tables.

If you change a type from `NOT FINAL` to `FINAL`, then you must specify `CASCADE [INCLUDING TABLE`

DATA]. You cannot defer data conversion with CASCADE NOT INCLUDING TABLE DATA.

If you change a type from FINAL to NOT FINAL:

Specify CASCADE INCLUDING TABLE DATA if you want to create new substitutable tables and columns of that type, but you are not concerned about the substitutability of the existing dependent tables and columns. Oracle Database marks all existing dependent columns and tables NOT SUBSTITUTABLE AT ALL LEVELS, so you cannot insert the new subtype instances of the altered type into these existing columns and tables.

Specify CASCADE CONVERT TO SUBSTITUTABLE if you want to create new substitutable tables and columns of the type and also store new subtype instances of the altered type in existing dependent tables and columns. Oracle Database marks all existing dependent columns and tables SUBSTITUTABLE AT ALL LEVELS except those that are explicitly marked NOT SUBSTITUTABLE AT ALL LEVELS.

Restriction on FINAL

You cannot change a user-defined type from NOT FINAL to FINAL if the type has any subtypes.

[NOT] INSTANTIABLE

Use this clause to indicate whether any object instances of this type can be constructed:

Specify INSTANTIABLE if object instances of this type can be constructed.

Specify NOT INSTANTIABLE if no constructor (default or user-defined) exists for this object type. You must specify these keywords for any type with noninstantiable methods and for any type that has no attributes (either inherited or specified in this statement).

Restriction on NOT INSTANTIABLE

You cannot change a user-defined type from INSTANTIABLE to NOT INSTANTIABLE if the type has any table dependents.

alter_collection_clauses

These clauses are valid only for collection types.

MODIFY LIMIT *integer*

This clause lets you increase the number of elements in a varray. It is not valid for nested tables. Specify an integer greater than the current maximum number of elements in the varray.

ELEMENT TYPE *datatype*

This clause lets you increase the precision, size, or length of a scalar datatype of a varray or nested table. This clause is not valid for collections of object types.

For a collection of NUMBER, you can increase the precision or scale.

For a collection of RAW, you can increase the maximum size.

For a collection of VARCHAR2 or NVARCHAR2, you can increase the maximum length.

dependent_handling_clause

The *dependent_handling_clause* lets you instruct Oracle Database how to handle objects that are dependent on the modified type. If you omit this clause, then the ALTER TYPE statement will abort if *type* has any dependent type or table.

INVALIDATE Clause

Specify INVALIDATE to invalidate all dependent objects without any checking mechanism.

Note:

Oracle Database does not validate the type change, so you should use this clause with caution. For example, if you drop an attribute that is a partitioning or cluster key, then you will be unable to write to the table.

CASCADE Clause

Specify the CASCADE clause if you want to propagate the type change to dependent types and tables. Oracle Database aborts the statement if any errors are found in the dependent types or tables unless you also specify FORCE.

If you change the property of the type between FINAL and NOT FINAL, you must specify this clause to convert data in dependent columns and tables. Please refer to [NOT] FINAL.

INCLUDING TABLE DATA

Specify INCLUDING TABLE DATA to convert data stored in all user-defined columns to the most recent version of the column type. This is the default.

Note:

You must specify this clause if your column data is in Oracle8 release 8.0 image format. This clause is also required if you are changing the type property between FINAL and NOT FINAL

For each attribute added to the column type, Oracle Database adds a new attribute to the data and initializes it to null.

For each attribute dropped from the referenced type, Oracle Database removes the corresponding attribute data from each row in the table.

If you specify `INCLUDING TABLE DATA`, then all of the tablespaces containing the table data must be in read/write mode.

If you specify `NOT INCLUDING TABLE DATA`, then the database upgrades the metadata of the column to reflect the changes to the type but does not scan the dependent column and update the data as part of this `ALTER TYPE` statement. However, the dependent column data remains accessible, and the results of subsequent queries of the data will reflect the type modifications.

CONVERT TO SUBSTITUTABLE

Specify this clause if you are changing the type from `FINAL` to `NOT FINAL` and you want to create new substitutable tables and columns of the type and also store new subtype instances of the altered type in existing dependent tables and columns. See [NOT] FINAL for more information.

FORCE

Specify `FORCE` if you want Oracle Database to ignore the errors from dependent tables and indexes and log all errors in the specified exception table. The exception table must already have been created by executing the `DBMS_UTILITY.CREATE_ALTER_TYPE_ERROR_TABLE` procedure.

Examples

Adding a Member Function: Example

The following example uses the `data_typ` object type, which was created in "Object Type Examples". A method is added to `data_typ` and its type body is modified to correspond. The date formats are consistent with the `order_date` column of the `oe.orders` sample table:

```
ALTER TYPE data_typ
  ADD MEMBER FUNCTION qtr(der_qtr DATE)
  RETURN CHAR CASCADE;

CREATE OR REPLACE TYPE BODY data_typ IS
MEMBER FUNCTION prod (invent NUMBER) RETURN NUMBER IS
BEGIN
RETURN (year + invent);
END;
 MEMBER FUNCTION qtr(der_qtr DATE) RETURN CHAR IS
 BEGIN
  IF (der_qtr < TO_DATE('01-APR', 'DD-MON')) THEN
    RETURN 'FIRST';
  ELSIF (der_qtr < TO_DATE('01-JUL', 'DD-MON')) THEN
    RETURN 'SECOND';
  ELSIF (der_qtr < TO_DATE('01-OCT', 'DD-MON')) THEN
```

```
    RETURN 'THIRD';
   ELSE
    RETURN 'FOURTH';
   END IF;
  END;
END;
/
```

Adding a Collection Attribute: Example

The following example adds the `phone_list_typ` varray from the sample `oe` schema to the `cust_address_typ` object column of the `customers` table:

```
ALTER TYPE cust_address_typ
  ADD ATTRIBUTE (phone phone_list_typ) CASCADE;
```

Increasing the Number of Elements of a Collection Type: Example

The following example increases the maximum number of elements in the varray `phone_list_typ`:

```
ALTER TYPE phone_list_typ
 MODIFY LIMIT 1000 CASCADE;
```

Increasing the Length of a Collection Type: Example

The following example increases the length of the varray element type `phone_list_typ`:

```
ALTER TYPE phone_list_typ
 MODIFY ELEMENT TYPE VARCHAR(64) CASCADE;
```

Recompiling a Type: Example

The following example recompiles type `customer_address_typ`:

```
ALTER TYPE customer_address_typ COMPILE;
```

Recompiling a Type Specification: Example

The following example compiles the type specification of `link2`.

```
CREATE TYPE link1 AS OBJECT
 (a NUMBER);
/
```

```
CREATE TYPE link2 AS OBJECT
  (a NUMBER,
   b link1,
MEMBER FUNCTION p(c1 NUMBER) RETURN NUMBER);
/
CREATE TYPE BODY link2 AS
MEMBER FUNCTION p(c1 NUMBER) RETURN NUMBER IS
  BEGIN
    dbms_output.put_line(c1);
    RETURN c1;
  END;
  END;
/
```

In the following example, both the specification and body of `link2` are invalidated because `link1`, which is an attribute of `link2`, is altered.

```
ALTER TYPE link1 ADD ATTRIBUTE (b NUMBER) INVALIDATE;
```

You must recompile the type by recompiling the specification and body in separate statements:

```
ALTER TYPE link2 COMPILE SPECIFICATION;

ALTER TYPE link2 COMPILE BODY;
```

Alternatively, you can compile both specification and body at the same time:

```
ALTER TYPE link2 COMPILE;
```