10

Registering Application Data Usage with the Database

This chapter details how you can use centralized database-centric entities called data use case domains and schema annotations to register information about the intended application data usage.

Oracle Database 23ai introduces a centralized, database-centric approach to handling intended data usage information using data use case domains and schema annotations. You can add data use case domains and schema annotations centrally in the database to register data usage intent, which is then accessible to various applications and tools.

> **See Also:**
>
> - *Oracle Database Concepts* for information about Data Use Case Domains and Schema Annotations.
> - *Oracle Database SQL Language Reference* for the syntactic and semantic information about data use case domains and schema annotations.

Sections:

10.1 Data Use Case Domains

This section explains how you can use data use case domains (hereinafter "use case domains") in your applications.

Topics:

- Viewing Domain Information
- Built-in Use Case Domains

10.1.1 Overview of Use Case Domains

Use case domains are lightweight usage specifiers with the optional database-side enforcement that applications can use to centrally document intended data usage. As a high-level dictionary object, a use case domain includes built-in usage properties associated with table columns, such as the default value, check constraints, collations, display and order formats, and annotations. Using centralized domain information, applications can standardize operations without depending on application-level metadata. For example, you can use use case domains to mask credit card numbers or to format phone numbers and currency values.

As a database object, a use case domain belongs to a schema and provides a common column definition without modifying the underlying column data type. A use case domain encapsulates some common characteristics of a table column into a reusable object that can be reused on other table columns without having to repeat these characteristics.

For example, a schema may have many tables with columns that hold email addresses, such as billing emails, invoice emails, and customer contact emails. Email addresses have a special format because they require the "@" sign. You can define a use case domain for the email address using a check constraint such as `regexp_like (email_dom, '^(\S+)\@(\S+)\.(\S+)$')`, and associate the domain with other email columns to have your application display the associated column's email addresses with the standard "@" sign prior to the domain name. Likewise, you may want to standardize display formats for vehicle license plates, which may require hyphens to separate the number from the other information. Using a use case domain, you can show the license information as "ABC-123" even when the license column has a varchar2(6) data type.

Benefits of using a use case domain include: improved data quality because it enables you to handle values consistently across applications; reduced coding because you can reuse a column's properties and constraints across your application; and consistency and clarity in operations because applications can inquire about these use case domains to understand what data they are operating against.

> **See Also:**
>
> *Oracle Database Concepts* for more information about Use Case Domains.

10.1.2 Use Case Domain Types and When to Use Them

There are three different types of domains: single-column, multi-column, and flexible domains.

Single-column Use Case Domains

A single-column use case domain is created for one column, and is used when you want to make a column usage definition consistent throughout your application. For example, you can create a single-column domain for email addresses, postal codes, or vehicle numbers.

Unless prefixed with multi-column or flexible, a "use case domain" or "domain" means a single-column use case domain for the purposes of this document.

Multi-column Use Case Domains

A multi-column use case domain creates column usage definitions for multiple columns under a single domain. A multi-column use case domain is ideal when you have logical entities spanning across two or more table columns. For example, you can use a multi-column use case domain to represent an address.

Flexible Use Case Domains

A flexible use case domain is a domain that dynamically selects from a set of single-column or multi-column use case domains based on a discriminant column's values. A flexible domain assigns a specific domain (also known as a constituent domain) to a set of value columns based on a mapping expression composed from one or more discriminant columns.

In addition to these use case domain types, you can also use built-in use case domains directly on table columns. Some examples are `email_d`, `ssn_d`, and `credit_card_number_d`.

See Also:

- *Oracle Database Concepts* for more information about types of Data Use Case Domains.
- Built-in Use Case Domains

10.1.3 Privileges Required for Use Case Domains

To work with use case domains, you require the following privileges:

Note:

The Database Administrator (DBA) role includes all the following privileges.

DDL Privilege	Action
CREATE DOMAIN	You can create a domain in your own schema. The RESOURCE and DB_DEVELOPER_ROLE roles include the CREATE DOMAIN privilege.
CREATE ANY DOMAIN	You can create a domain in any schema.
ALTER ANY DOMAIN	You can alter a domain in any schema.
DROP ANY DOMAIN	You can drop a domain in any schema.
EXECUTE ANY DOMAIN	You can reference or use a domain in any schema. To explicitly grant execute privileges to a user on a domain in any schema, use the following code: `GRANT EXECUTE ON <schemaName.domainName> TO <user>;`

10.1.4 Using a Single-column Use Case Domain

This section explains how you can create, associate, alter, disassociate, and drop a single-column use case domain (hereinafter "use case domain" or "domain").

Topics

10.1.4.1 Creating a Use Case Domain

You can define a use case domain that encapsulates a set of optional properties and constraints to represent common values, such as email addresses and credit card numbers.

The following are some examples of creating use case domains, where a domain is based on the properties of a table column, such as default value, constraints, annotations, or display and order expressions.

Example 10-1 Creating a Use Case Domain for Hourly Wages

A Human Resource Application (HRA) creates many tables for different companies. Most companies have a column that stores hourly wages. The HRA can create a use case domain for the hourly wages columns as follows:

```
CREATE DOMAIN HourlyWages AS NUMBER
  DEFAULT ON NULL 15
  CONSTRAINT MinimalWage CHECK (HourlyWages > = 7 AND HourlyWages <=1000)
ENABLE
  DISPLAY TO_CHAR(HourlyWages, '$999.99')  ORDER ( -1*HourlyWages )
  ANNOTATIONS (properties '{"Purpose": "Wages", "Applicability": "USA",
"Industry": {"Sales", "Manufacturing"} }');
```

Example 10-2 Creating a Use Case Domain for Surrogate Keys

If you want to annotate surrogate key columns in your database application and maintain standard meta-data for such columns, create a domain similar to the following:

```
CREATE DOMAIN surrogate_id AS INTEGER
  STRICT
  NOT NULL
  ANNOTATIONS ( primary_key, mandatory, operations '["insert", "delete"]' );
```

Example 10-3 Creating a Use Case Domain for Birth Dates

The following is a use case domain that ensures that all columns with birth dates are "date only" and displays the age in years.

```
CREATE DOMAIN birth_date AS DATE
  CONSTRAINT birth_date_only_c check ( birth_date = trunc ( birth_date ) )
  DISPLAY FLOOR ( months_between ( sysdate, birth_date ) / 12 ) || ' years'
  ANNOTATIONS ( sensitive 'PII Data', operations '["insert", "update"]' );
```

Example 10-4 Creating a Use Case Domain for Default Date and Time Values

You can create a use case domain for date and time values to ensure that the inserts are in a standard format.

```
CREATE DOMAIN insert_timestamp AS
  TIMESTAMP WITH LOCAL TIME ZONE
  DEFAULT ON NULL systimestamp;
```

Example 10-5 Creating a Use Case Domain for Positive Heights

The following use case domain ensures that people have positive heights and the heights are sorted in a descending order.

```
CREATE DOMAIN height AS NUMBER
  CONSTRAINT positive_height_c CHECK ( value > 0 )
  ORDER value * -1
  ANNOTATIONS ( operations '["insert", "update"]' );
```

Example 10-6 Creating a Use Case Domain for Positive Weights

The following use case domain ensures that people have positive weights.

```
CREATE DOMAIN weight AS NUMBER
  CONSTRAINT positive_weight_c CHECK ( value > 0 )
  ANNOTATIONS ( operations '["insert", "update"]' );
```

Example 10-7 Defining a Domain with Multiple Check Constraints

You can define a use case domain for email addresses, similar to the one in the following example, which also has multiple `CHECK` constraints.

```
CREATE SEQUENCE IF NOT EXISTS email_seq;

CREATE DOMAIN email AS VARCHAR2(100)
  DEFAULT ON NULL email_seq.NEXTVAL || '@domain.com'
  CONSTRAINT email_c CHECK (REGEXP_LIKE (email, '^(\S+)\@(\S+)\.(\S+)$'))
  CONSTRAINT email_max_len_c CHECK (LENGTH(email) <=100) DEFERRABLE INITIALLY
DEFERRED
  DISPLAY '---' || SUBSTR(email, INSTR(email, '@') + 1);
```

Any column of `VARCHAR2(L [BYTE|CHAR])` data type that satisfies both constraints can be associated with the domain. The `INITIALLY DEFERRED` clause delays validation of the constraint `email_max_len_c` values until commit time.

Example 10-8 JSON Schema Validation

A domain can also be used for reusable JSON schema validation, as in the following example.

```
CREATE DOMAIN department_json_doc AS JSON
  CONSTRAINT CHECK (
    department_json_doc IS JSON VALIDATE USING '{
      "type": "object",
      "properties": {
        "departmentName": { "type": "string" },
        "employees": { "type": "array" }
      },
      "required" : [ "departmentName", "employees" ],
      "additionalProperties": false
    }' );
```

See Also:

- *Oracle Database SQL Language Reference* for the syntactic and semantic information on creating a use case domain: CREATE DOMAIN
- Specifying a Data Type for a Domain

10.1.4.2 Associating Use Case Domains with Columns at Table Creation

After defining a use case domain, you can associate the domain with new table columns when creating a table or associate it with existing columns in an existing table. Associating a use case domain with a column explicitly applies the domain's optional properties and constraints to the column.

You can use the `CREATE TABLE` DDL to associate a use case domain with a newly created column. The following examples show how you can associate a use case domain with columns in a new table. These examples use the use case domains created in the earlier examples.

Example 10-9 Associating `HourlyWages` Use Case Domain at Table Creation

Using the `HourlyWages` domain, the HRA can create multiple tables where wage columns have the same domain characteristics.

```
CREATE TABLE employee (
  name VARCHAR2(100),
  id NUMBER,
  wage NUMBER DOMAIN HourlyWages);


CREATE TABLE wage (
  name VARCHAR2(100),
  id NUMBER,
  wage NUMBER DOMAIN HourlyWages,
  gross_pay NUMBER,
  deductions NUMBER,
  net_pay NUMBER);
```

Example 10-10 Associating the surrogate_id Use Case Domain at Table Creation

When you associate a strict use case domain such as surrogate_id with columns, ensure that the associated columns have the same data type as the domain. A strict domain also requires that the column length, scale, and precision match with the domain.

The following code fails with ORA-11517: the column data type does not match the domain column because you are trying to link a NUMBER data type column with a INTEGER/NUMBER(*,0) data type domain.

```
CREATE TABLE orders (
  order_id NUMBER DOMAIN surrogate_id,
  customer_id NUMBER,
  order_datetime TIMESTAMP WITH LOCAL TIME ZONE
    DEFAULT SYSTIMESTAMP);
```

To ensure that the association works, you can use the NUMBER(*,0) column data type to associate with the surrogate_id use case domain (INTEGER == NUMBER(*,0)).

```
CREATE TABLE orders (
  order_id NUMBER(*,0) DOMAIN surrogate_id,
  customer_id NUMBER,
  order_datetime TIMESTAMP WITH LOCAL TIME ZONE
    DEFAULT SYSTIMESTAMP);
```

Example 10-11 Associating the surrogate_id, birth_date, height, and weight Use Case Domains at Table Creation

The DOMAIN keyword is optional. You can see in the following example that the birth_date domain is associated with date_of_birth column without the DOMAIN keyword. The example also shows that you can define a more precise data type for the columns with regards to the precision and scale than what is in the domain. The height_in_cm and weight_in_kg columns have their associated domain data type as NUMBER whereas the column data type has precision and scale values, such as NUMBER(4,1).

```
CREATE TABLE people (
  person_id     DOMAIN surrogate_id
    GENERATED BY DEFAULT AS IDENTITY
    PRIMARY KEY,
  full_name     VARCHAR2(256),
  date_of_birth birth_date,
  height_in_cm  NUMBER(4, 1) DOMAIN height,
  weight_in_kg  NUMBER(6, 3) DOMAIN weight);
```

Example 10-12 Associating the department_json_doc Use Case Domain at Table Creation

```
CREATE TABLE departments (
  department_id  INTEGER PRIMARY KEY,
  department_doc JSON DOMAIN department_json_doc);
```

Guidelines

- When associating a domain with a column, you can specify the domain name in addition to the column's data type, in which case the column's data type is used, provided that the domain data type is compatible with the column's data type.
- When associating a domain with a column, you can specify a domain name instead of the column's data type, in which case the domain data type is used for the column, wherein the `DOMAIN` keyword is optional.
- If a domain is defined as `STRICT`, the domain's data type, scale, and precision must match the column's data type, scale, and precision.
- If a domain is not defined as `STRICT`, you can associate a domain of any length with a column of any length. For instance, you can associate a domain of `VARCHAR2(10)` with any `VARCHAR2` column.

> **See Also:**
>
> - *Oracle Database SQL Language Reference* for the syntactic and semantic information about creating a use case domain: CREATE DOMAIN
> - Specifying a Data Type for a Domain for information about column and domain data types

10.1.4.2.1 Using DML on Columns Associated with Use Case Domains

The following are some examples of DML statements that you can use on the newly created table columns with associated use case domains.

Example 10-13 Using DML Statements on the `people` table

The following `INSERT` commands insert data into the `people` table columns, while verifying that the check constraint specified in the associated domain is not violated.

```
INSERT INTO people
  VALUES ( 1, 'Sally Squirell', date'1981-01-01', 180.1, 61 );

INSERT INTO people
  VALUES ( 2, 'Brian Badger', date'2016-12-31', 120.4, 27.181 );
```

The following `INSERT` command fails because `height` is specified as a negative number, and hence the associated check constraint is violated.

```
INSERT INTO people
  VALUES ( 3, 'Fergal Fox', date'2023-04-12', -99, 1 );
```

The output is:

```
ORA-11534: check constraint (HR.SYS_C009232) due to domain constraint
HR.POSITIVE_HEIGHT_C of domain HR.HEIGHT violated
```

You can use the associated domains to display data with the heights sorted in a descending order and also view the corresponding age and weight.

```
SELECT full_name, DOMAIN_DISPLAY ( date_of_birth ) age,
       height_in_cm, weight_in_kg
  FROM people
  ORDER BY DOMAIN_ORDER ( height_in_cm );
```

> **See Also:**
>
> SQL Functions for Use Case Domains for details about domain functions, such as `DOMAIN_DISPLAY` and `DOMAIN_ORDER`.

The output is:

```
FULL_NAME            AGE        HEIGHT_IN_CM WEIGHT_IN_KG
-------------------- ---------- ------------ ------------
Sally Squirell       42 years          180.1           61
Brian Badger         6 years           120.4       27.181
```

You can describe the `people` table to see the data type definition and the referenced domain information.

```
DESC people;
```

The output is:

```
 Name                  Null?    Type
 --------------------- -------- ----------------------------------------
 PERSON_ID             NOT NULL NUMBER(38) HR.SURROGATE_ID
 FULL_NAME                      VARCHAR2(256)
 DATE_OF_BIRTH                  DATE HR.BIRTH_DATE
 HEIGHT_IN_CM                   NUMBER(4,1) HR.HEIGHT
 WEIGHT_IN_KG                   NUMBER(6,3) HR.WEIGHT
```

The following `SELECT` command enables you to view the column annotations that are inherited from the associated domains.

```
SELECT column_name, annotation_name, annotation_value
  FROM user_annotations_usage
  WHERE object_name = 'PEOPLE';
```

The output is:

```
COLUMN_NAME          ANNOTATION_NAME      ANNOTATION_VALUE
-------------------- -------------------- --------------------
PERSON_ID            PRIMARY_KEY          <null>
PERSON_ID            MANDATORY            <null>
PERSON_ID            OPERATIONS           ["insert", "delete"]
DATE_OF_BIRTH        OPERATIONS           ["insert", "update"]
DATE_OF_BIRTH        SENSITIVE            PII Data
HEIGHT_IN_CM         OPERATIONS           ["insert", "update"]
WEIGHT_IN_KG         OPERATIONS           ["insert", "update"]
```

Example 10-14 Using DML Statements on the `departments` table with JSON data

The following `INSERT` command on the `departments` table succeeds because it includes all the JSON attributes in the `department_json_doc` domain.

```
INSERT INTO departments
  VALUES ( 1, '{
    "departmentName" : "Accounting",
    "employees" : [
      {"empName":"William"},
      {"empName":"Shelley"}
    ]
  }');
```

The following `INSERT` command fails with `ORA-40875: JSON schema validation error - missing employees attribute` because the `employees` attribute is missing.

```
INSERT INTO departments
  VALUES ( 2, '{
    "departmentName" : "Finance"
  }');
```

The following `INSERT` command fails with `ORA-40875: JSON schema validation error - extra manager attribute` because the `manager` attribute is not found in the associated `department_json_doc` use case domain.

```
INSERT INTO departments
  VALUES ( 3, '{
    "departmentName" : "Executive",
    "employees" : [
      {"empName":"Lex"},
      {"empName":"Neena"}
    ],
    "manager" : {"empName":"Lex"}
  }');
```

10.1.4.3 Associating Use Case Domains with Existing or New Columns

You can use the `ALTER TABLE` DDL with the `MODIFY` or `ADD` clause to associate a use case domain with an existing or a newly added column.

Example 10-15 Associating a Use Case Domain with a Newly Created Column

For a newly created `customers` table:

```
CREATE TABLE customers (
  cust_id NUMBER,
  cust_email VARCHAR2(100));
```

You can use `ADD` to add a new column: `cust_new_email` and associate it with the `email` domain.

```
ALTER TABLE customers
  ADD (cust_new_email VARCHAR2(100) DOMAIN email);
```

Example 10-16 Associating the email Use Case Domain with an Existing Column

You can use `MODIFY` to modify an existing column: `cust_email` and associate it with the `email` domain.

```
ALTER TABLE customers
  MODIFY (cust_email  VARCHAR2(100) DOMAIN email);
```

You can also add a use case domain to a column using the `ALTER TABLE ... MODIFY` statement. In the following example, the `orders` table column, namely `order_datetime` and the `insert_timestamp` domain have different defaults. The `insert_timestamp` domain has the `DEFAULT` with `ON NULL` clause, which is missing in the `order_datetime` column. Therefore, when you try to associate the domain with the column, you get an error `ORA-11501: The column default does not match the domain default of the column.`

```
ALTER TABLE orders
  MODIFY order_datetime DOMAIN insert_timestamp;
```

To overcome the default mismatch, specify a column default clause that matches the domain default.

```
ALTER TABLE orders
  MODIFY order_datetime DOMAIN insert_timestamp
    DEFAULT ON NULL systimestamp;
```

Example 10-17 Querying to View Associated Domains

```
SELECT constraint_name, search_condition_vc, domain_constraint_name
  FROM  user_constraints
  JOIN  user_cons_columns
  USING ( constraint_name, table_name )
  WHERE  table_name = 'PEOPLE'
    AND  constraint_type = 'C'
    AND  column_name = 'WEIGHT_IN_KG';
```

The output is:

```
CONSTRAINT_NAME      SEARCH_CONDITION_VC            DOMAIN_CONSTRAINT_NAME
-------------------- ------------------------------ -------------------------
SYS_C008493          "WEIGHT_IN_KG">0               POSITIVE_WEIGHT_C
```

Guidelines

- The `DOMAIN` keyword is optional for the `ALTER TABLE .. ADD` statement if a domain only is specified for the newly added column.
- The `DOMAIN` keyword is mandatory for the `ALTER TABLE .. MODIFY` statement.
- The column data type should be compatible with the domain data type.
- If a domain has default expression or collation, it should match the associated column's default expression and collation.
- If the associated column already has a domain associated with it, an error is returned.

10.1.4.4 Altering a Use Case Domain

You can alter a use case domain for its display expression, order expression, and annotation. The following `ALTER DOMAIN` DDL statements are supported.

Example 10-18 Removing the Order Expression from a Domain

The `height` use case domain definition has the order expression as `VALUE`, so to remove the order expression, you must drop the order expression.

```
ALTER DOMAIN height DROP ORDER;
```

Example 10-19 Adding a Display Expression to a Use Case Domain

To add a display expression to the `height` use case domain, use the following `ALTER` command.

```
ALTER DOMAIN height
  ADD DISPLAY round ( value ) || ' cm';
```

Example 10-20 Changing the Display Expression of the `birth_date` Domain

To change the display expression of the `birth_date` domain to years and months, use the following `ALTER` command.

```
ALTER DOMAIN birth_date
  MODIFY DISPLAY
    FLOOR ( months_between ( sysdate, birth_date ) / 12 ) || ' years ' ||
    MOD ( FLOOR ( months_between ( sysdate, birth_date ) ), 12 ) || ' months';
```

Example 10-21 Querying the `people` table for the Altered `birth_date`, `height`, and `weight` Domains

```
COLUMN age FORMAT A20

SELECT full_name, DOMAIN_DISPLAY ( date_of_birth ) age,
```

```
      DOMAIN_DISPLAY ( height_in_cm ) height_in_cm, weight_in_kg
  FROM  people
  ORDER  BY DOMAIN_ORDER ( height_in_cm );
```

> **See Also:**
>
> SQL Functions for Use Case Domains for details about domain functions, such as `DOMAIN_DISPLAY` and `DOMAIN_ORDER`.

The output is:

```
FULL_NAME            AGE                  HEIGHT_IN_CM  WEIGHT_IN_KG
-------------------- -------------------- ------------- ------------
Sally Squirell       42 years 5 months    180 cm                  61
Brian Badger         6 years 5 months     120 cm               27.181
```

Example 10-22 Changing the Annotations on a Use Case Domain

The following code adds the annotations for the `height` use case domain.

```
ALTER DOMAIN height
  ANNOTATIONS (
    operations '["insert", "update", "sort"]',
    sensitive 'Private data');
```

Example 10-23 Querying the Dictionary Views for the Annotation Changes

```
COLUMN annotation_value FORMAT A40

SELECT column_name, annotation_name, annotation_value
  FROM user_annotations_usage
  WHERE object_name = 'PEOPLE';
```

The output is:

```
COLUMN_NAME          ANNOTATION_NAME
ANNOTATION_VALUE
-------------------- --------------------
----------------------------------------
PERSON_ID            PRIMARY_KEY
<null>
PERSON_ID            MANDATORY
<null>
PERSON_ID            OPERATIONS           ["insert",
"delete"]
DATE_OF_BIRTH        SENSITIVE            PII
Data
DATE_OF_BIRTH        OPERATIONS           ["insert",
"update"]
```

```
HEIGHT_IN_CM          OPERATIONS            ["insert",
"update"]
HEIGHT_IN_CM          OPERATIONS            ["insert", "update",
"sort"]
HEIGHT_IN_CM          SENSITIVE             Private
data
WEIGHT_IN_KG          OPERATIONS            ["insert",
"update"]
```

Guidelines

- You can alter the display expression of a domain only if the domain is not constituent in a flexible domain.
- You can alter the order expression of a domain only if the domain is not constituent in a flexible domain.
- You can alter only domain-level annotations.

> **See Also:**
>
> - *Oracle Database SQL Language Reference* for the syntactic and semantic information about altering a use case domain: ALTER DOMAIN.
> - Viewing Domain Information for the use case domain dictionary views

10.1.4.5 Disassociating a Use Case Domain from a Column

You can use the `ALTER TABLE` DDL with the `DROP` clause to disassociate a use case domain from a column.

Example 10-24 Disassociating a Use Case Domain from a Column

To drop the associated domain from the `cust_email` column of the `customers` table:

```
ALTER TABLE customers
  MODIFY ( cust_email ) DROP DOMAIN;
```

To drop the domain but keep the domain's constraint:

```
ALTER TABLE customers
  MODIFY ( cust_email ) DROP DOMAIN PRESERVE CONSTRAINTS;
```

Example 10-25 Disassociating a Use Case Domain from a Column

The following code removes the domain association of the `height` domain from the `height_in_cm` column in the `people` table, while preserving the constraint.

```
ALTER TABLE people
  MODIFY ( height_in_cm ) DROP DOMAIN PRESERVE CONSTRAINTS;
```

Example 10-26 Querying the Columns with Disassociated Use Case Domain

The `SELECT` query on the `user_constraints` dictionary table reveals the changes.

```
SELECT constraint_name, search_condition_vc, domain_constraint_name
  FROM user_constraints
  JOIN user_cons_columns
  USING ( constraint_name, table_name )
  WHERE table_name = 'PEOPLE'
    AND constraint_type = 'C';
```

The output is:

```
CONSTRAINT_NAME      SEARCH_CONDITION_VC
DOMAIN_CONSTRAINT_NAME
-------------------- ----------------------------------------
------------------------------
SYS_C009491          "DATE_OF_BIRTH"=TRUNC("DATE_OF_BIRTH")
BIRTH_DATE_ONLY_C
SYS_C009494          "WEIGHT_IN_KG">0
POSITIVE_WEIGHT_C
SYS_C009489          "PERSON_ID" IS NOT NULL
<null>
SYS_C009490          "HEIGHT_IN_CM">0
<null>
```

Example 10-27 Re-adding the Removed `height` Domain

The following code re-adds the removed `height` domain to the `height_in_cm` column in the `people` table.

```
ALTER TABLE people
  MODIFY ( height_in_cm ) ADD DOMAIN height;
```

Example 10-28 Querying the Re-added Domain

A duplicate `height_in_cm > 0` constraint is created with the re-added `height` domain.

```
SELECT constraint_name, search_condition_vc, domain_constraint_name
  FROM user_constraints
  JOIN user_cons_columns
  USING ( constraint_name, table_name )
  WHERE table_name = 'PEOPLE'
    AND constraint_type = 'C'
  ORDER BY search_condition_vc;
```

The output is:

```
CONSTRAINT_NAME      SEARCH_CONDITION_VC
DOMAIN_CONSTRAINT_NAME
-------------------- ----------------------------------------
------------------------------
```

```
SYS_C009491          "DATE_OF_BIRTH"=TRUNC("DATE_OF_BIRTH")
BIRTH_DATE_ONLY_C
SYS_C009495          "HEIGHT_IN_CM">0
POSITIVE_HEIGHT_C
SYS_C009490          "HEIGHT_IN_CM">0
<null>
SYS_C009489          "PERSON_ID" IS NOT NULL
<null>
SYS_C009494          "WEIGHT_IN_KG">0
POSITIVE_WEIGHT_C
```

Guidelines

On dropping the domain for a column, the following are preserved by default:

- The domain's collation.
- The non-domain constraint that is added to the column.

The domain's default value is not preserved. It is only kept if the default is explicitly applied to the column.

> **See Also:**
>
> - *Oracle Database SQL Language Reference* for the syntactic and semantic information about altering a use case domain: ALTER DOMAIN.
> - Viewing Domain Information for the use case domain dictionary views

10.1.4.6 Dropping a Use Case Domain

You can drop a use case domain. See *Oracle Database SQL Language Reference* for more information about dropping a domain.

Example 10-29 Dropping a Use Case Domain

The following `customers` table has one of its column associated with the `email` domain:

```
CREATE TABLE customers (
  cust_id NUMBER,
  cust_email VARCHAR2(100) DOMAIN email);
```

The following `DROP` command returns an error because the `customers` table has the `cust_email` column associated with `email` domain.

```
DROP DOMAIN email;
```

The following `DROP` command succeeds:

```
DROP DOMAIN email FORCE;
```

The `cust_email` column is disassociated from the `email` domain and all statements mentioning the `email` domain are invalidated.

Example 10-30 Dropping a Domain but Preserving the Constraint

The following `DROP` command succeeds but preserves the default and constraint expressions.

```
DROP DOMAIN email FORCE PRESERVE;
```

The `cust_email` column retains the default `ON NULL email_seq.NEXTVAL || '@domain.com'` and preserves the constraint after replacing the domain name in the constraint with the column name: `CONSTRAINT email_c CHECK (REGEXP_LIKE (cust_email, '^(\S+)\@(\S+)\.(\S+)$'))`.

Guidelines

- To drop a domain that is referenced in a flexible domain, use `DROP DOMAIN` with the `FORCE` option.
- If the domain is not associated with any table column and the domain is not constituent in a flexible domain, the domain is dropped. If the use case domain is in use, the `DROP` statement fails.
- If the domain is associated with any table column, you must use the `FORCE` option to drop the domain. Using the `FORCE` option also:
  - Removes the default expression, if only domain default is set.
  - Preserves the column default expression, if both domain and column defaults are set.
  - Removes the domain annotation from all associated columns.
  - Preserves the collation of any domain associated columns.
  - Invalidates all SQL dependent statements in the cursor cache.
  - Preserves the constraints on any domain associated columns, if `FORCE PRESERVE` is used.

See Also:

- *Oracle Database SQL Language Reference* for the syntactic and semantic information about dropping a use case domain: DROP DOMAIN.
- About Drop Domain and Recycle Bin

10.1.4.6.1 About Drop Domain and Recycle Bin

The dropped tables are placed in the recycle bin. To drop a domain associated with tables in the recycle bin, you must use the `FORCE` option with the `DROP` command.

You can restore the dropped tables that are in the recycle bin to the before-the-drop position using the `FLASHBACK TABLE` command. If a table is restored to the before-the-drop position (using `FLASHBACK TABLE TO BEFORE DROP`) after the associated domain has been dropped, then the table has the same default, collation, nullability, and constraints as was there before the drop, except that none of these attributes would be marked as being inherited from the domain.

Example 10-31 Dropping a Use Case Domain Associated with a Dropped Table

The following `DROP` command tries to remove the `weight` domain that is associated with the `weight_in_kg` column of the `people` table.

```
DROP DOMAIN weight;
```

The command returns the following error output:

```
ORA-11502: The domain WEIGHT to be dropped has dependent objects.
```

If you drop the `people` table, and then drop the weight domain, it returns an error because the table is still in the recycle bin.

```
DROP TABLE people;
DROP DOMAIN weight;
```

The command returns the following error output:

```
ORA-11502: The domain WEIGHT to be dropped has dependent objects.
```

Removing the `people` table from the recycle bin permanently, and then running the `DROP` command on the `weight` domain, drops the `weight` domain.

```
PURGE TABLE people;
DROP DOMAIN weight;
```

Guidelines

Here are some points to note when dropping domains that are associated with dropped tables (tables in recycle bin):

- While a table is in the recycle bin, the `ALTER` command on the table is not allowed.
- If a table with a domain association is in the recycle bin, the associated domain cannot be dropped and the `DROP DOMAIN` command fails.
- When the `DROP DOMAIN FORCE` and `DROP DOMAIN FORCE PRESERVE` commands are used, the tables in the recycle bin are disassociated from the domain. The database uses the `FORCE PRESERVE` semantics for tables in the recycle bin, even if you only specify `FORCE`.
- If you want to drop the domain that is associated with a table in the recycle bin, you can use the `PURGE TABLE` command to remove a table from the recycle bin and run the `DROP DOMAIN` command to drop the domain.

10.1.5 Using a Multi-column Use Case Domain

This section explains how you can create, associate, alter, disassociate, and drop a multi-column use case domain.

Topics

- Creating a Multi-column Use Case Domain
- Associating a Multi-column Use Case Domain at Table Creation
- Associating a Multi-column Use Case Domain with Existing Columns
- Altering a Multi-column Use Case Domain
- Disassociating a Multi-column Use Case Domain from a Column
- Dropping a Multi-column Use Case Domain

10.1.5.1 Creating a Multi-column Use Case Domain

You can use a multi-column use case domain to group logical entities that span across table columns, such as addresses.

Example 10-32 Creating a Multi-column Use Case Domain for Addresses

You can create a multi-column use case domain called "US_city" with three columns for address entries, as follows:

```
CREATE DOMAIN US_city AS (
    name  AS VARCHAR2(30) ANNOTATIONS (Address),
    state AS VARCHAR2(2) ANNOTATIONS (Address),
    zip AS NUMBER ANNOTATIONS (Address)
  )
  CONSTRAINT City_CK CHECK(state in ('CA','AZ','TX') and zip < 100000)
  DISPLAY name||', '|| state ||', '||TO_CHAR(zip)
  ORDER state||', '||TO_CHAR(zip)||', '||name
  ANNOTATIONS (Title 'Domain Annotation');
```

Example 10-33 Creating a Multi-column Use Case Domain for Currency

The following code creates a multi-column use case domain called `currency` that displays monetary values as an amount in a given currency, and the currency codes. The display is sorted by value (low to high), and then by the currency code.

```
CREATE DOMAIN currency AS (
    amount  AS NUMBER,
    iso_currency_code  AS CHAR(3 CHAR)
  )
  DISPLAY iso_currency_code || TO_CHAR ( amount, '999,999,990.00' )
  ORDER TO_CHAR ( amount, '999,999,990.00' ) || iso_currency_code;
```

Guidelines

- You can have the same data types for the individual columns in a multi-column use case domain as a single-column use case domain.
- For a multi-column use case domain, a column must not overlap between different domains. For example, on a table T(TC1, TC2, TC3, TC4), domains D1(C1, C2) and D2(C1, C2) cannot be associated as D1(TC1, TC2) and D2(TC2, TC3).
- Multiple ordered subsets of columns in the same table can be associated with the same domain. For example, domain D1 can be associated as D1(TC1, TC2) and D1(TC3, TC4).

- Unlike tables that can have at most one `LONG` column, domains can have multiple columns of `LONG` data type. Such domains would be useful for evaluating check conditions involving multiple `LONG` columns using the `DOMAIN_CHECK` operator.

> **See Also:**
>
> - *Oracle Database SQL Language Reference* for the syntactic and semantic information on creating a use case domain: CREATE DOMAIN
> - Specifying a Data Type for a Domain

10.1.5.2 Associating a Multi-column Use Case Domain at Table Creation

You can use the `CREATE TABLE` DDL statement to associate a multi-column use case domain with a newly created column.

Example 10-34 Associating the `US_city` Domain with Multiple Columns

You can create a `customer` table and associate the `US_city` domain with the table's three columns.

```
CREATE TABLE customer(
  cust_id  NUMBER,
  cust_name  VARCHAR2(30),
  city_name  VARCHAR2(30),
  state  VARCHAR2(2),
  zip  NUMBER,
  DOMAIN US_city(city_name, state, zip));
```

The following example returns an error because `CITY` and `STATE` columns are overlapped between domains.

```
CREATE TABLE customer(
  cust_id  NUMBER,
  cust_name  VARCHAR2(30),
  city_name  VARCHAR2(30),
  state  VARCHAR2(2),
  zip  NUMBER,
  DOMAIN US_city(city_name, state, zip),
  DOMAIN US_city(cust_name, state, zip));
```

The following example also returns an error because the `CITY_NAME` column is repeated.

```
CREATE TABLE customer(
  cust_id  NUMBER,
  cust_name  VARCHAR2(30),
  city_name  VARCHAR2(30),
  state  VARCHAR2(2),
  zip  NUMBER,
  DOMAIN US_city(city_name, city_name, zip));
```

Example 10-35 Associating the `currency` Domain with Multiple Columns

You can create an `order_items` table with its `total_paid` and `currency_code` columns associated with the `currency` domain.

```
CREATE TABLE order_items (
  order_id INTEGER, product_id INTEGER,
  total_paid NUMBER(10, 2), currency_code char (3 CHAR ),
  DOMAIN currency ( total_paid, currency_code ));
```

Guidelines

- The column names that are passed as the actual parameters to the domain must be unique.
- Domain columns can be associated with table columns with a different name.
- The `DOMAIN` keyword is mandatory.

10.1.5.2.1 Using DML on Columns Associated with Multi-column Domains

The following are some examples of DML statements that you can use on the newly created table columns with associated multi-column use case domains.

Example 10-36 Using DML Commands on the Associated Columns

Inserting values and querying the table display the results based on the `currency` domain display and order expressions.

```
INSERT INTO order_items
VALUES (1, 1,    9.99, 'USD'),
       (2, 2,    8.99, 'GBP'),
       (3, 3,    8.99, 'EUR'),
       (4, 4, 1399,    'JPY'),
       (5, 5,  825,    'INR');

SELECT order_id, product_id,
  DOMAIN_DISPLAY ( total_paid, currency_code ) amount_paid
  FROM order_items
  ORDER BY DOMAIN_ORDER ( total_paid, currency_code );
```

> **See Also:**
>
> SQL Functions for Use Case Domains for details about domain functions, such as `DOMAIN_DISPLAY` and `DOMAIN_ORDER`.

The output is:

```
  ORDER_ID PRODUCT_ID AMOUNT_PAID
---------- ---------- ------------------
         3          3 EUR           8.99
         2          2 GBP           8.99
         1          1 USD           9.99
```

```
5          5 INR         825.00
4          4 JPY       1,399.00
```

10.1.5.3 Associating a Multi-column Use Case Domain with Existing Columns

You can use the `ALTER TABLE` DDL statement with the `MODIFY` or `ADD` clause to associate a multi-column use case domain with an existing column or a newly added column in an existing table.

Example 10-37 Associating a Multi-column Use Case Domain with Existing Columns

The following example applies the `US_city` domain to the three columns of the `customer` table.

```
ALTER TABLE customer
  MODIFY (city_name, state, zip) ADD DOMAIN US_city;
```

> **Note:**
>
> The `DOMAIN` keyword is mandatory for the `ALTER TABLE .. MODIFY` statement.

10.1.5.4 Altering a Multi-column Use Case Domain

You can alter a multi-column use case domain just as you can alter a single-column use case domain. In a multi-column use case domain, you can change the `DISPLAY` and `ORDER` properties. For multi-column domains, altering annotations at the column-level is currently not supported but you can alter the object-level annotations.

Example 10-38 Altering Display and Order Expressions for a Multi-column Use Case Domain

The following `ALTER` statement changes the display expression of the `currency` domain. The current display expression shows the currency code and then the currency value. The altered display expression shows the currency value and then the currency code.

```
ALTER DOMAIN currency
  MODIFY DISPLAY TO_CHAR ( amount, '999,990.00' ) || '-' || iso_currency_code;
```

The following `ALTER` statement changes the order expression of the `currency` domain. The current order expression sorts by the currency value and then by the currency code. The altered order expression sorts by the currency code and then by the currency value.

```
ALTER DOMAIN currency
  MODIFY ORDER iso_currency_code || TO_CHAR ( amount, '999,990.00' );
```

Example 10-39 Querying the Table Associated with the Altered Multi-column Domain

```
SELECT order_id, product_id,
  DOMAIN_DISPLAY ( total_paid, currency_code ) amount_paid
  FROM order_items
  ORDER BY DOMAIN_ORDER ( total_paid, currency_code );
```

The output is:

```
ORDER_ID    PRODUCT_ID    AMOUNT_PAID
----------- ------------- ---------------
          3             3        8.99-EUR
          2             2        8.99-GBP
          5             5      825.00-INR
          4             4    1,399.00-JPY
          1             1        9.99-USD
```

> **See Also:**
>
> - *Oracle Database SQL Language Reference* for the syntactic and semantic information about altering a use case domain: ALTER DOMAIN.
> - Altering a Use Case Domain

10.1.5.5 Disassociating a Multi-column Use Case Domain from a Column

You can use the `ALTER TABLE` DDL statement with the `DROP` clause to disassociate a multi-column use case domain from a column.

Example 10-40 Examples of Disassociating a Multi-column Use Case Domain

The following `ALTER TABLE` command drops the `US_City` domain from the `city_name`, `state` and `zip` columns of the `customer` table.

```
ALTER TABLE customer
  MODIFY(city_name, state, zip) DROP DOMAIN;
```

If a table T with columns (c1, c2, c3) is associated with domain D and another set of columns (c4, c5, c6) is also associated with the domain D, you can drop the domain for all the columns:

```
ALTER TABLE T
  MODIFY (c1, c2, c6, c5, c4, c3) DROP DOMAIN;
```

You cannot drop only a subset of the columns that are associated with a multi-column domain. For example, for table T, dropping only c1 and c2 columns, returns an error:

```
ALTER TABLE T
  MODIFY (c1, c2) DROP DOMAIN;
```

Example 10-41 More Examples of Disassociating a Multi-column Use Case Domain

The following code removes the `currency` domain from the `total_paid` and `currency_code` columns of the `order_items` table.

```
ALTER TABLE order_items
  MODIFY ( total_paid, currency_code ) DROP DOMAIN;
```

Guidelines

- There can be multiple ordered subsets of columns in the same table that are associated with the same domain. The removing multi-column domain syntax must specify the list of associated columns to be dissociated.
- Domain name cannot be specified.
- You cannot specify other options for `ALTER TABLE ..MODIFY` with `ALTER TABLE ..DROP DOMAIN`.

10.1.5.6 Dropping a Multi-column Use Case Domain

To drop a multi-column use case domain, use the same syntax as used for a single-column use case domain.

Guidelines

- To drop a domain that is referenced in a flexible domain, use `DROP DOMAIN` with the `FORCE` option.

> **See Also:**
>
> - *Oracle Database SQL Language Reference* for the syntactic and semantic information about dropping a use case domain: DROP DOMAIN
> - Dropping a Use Case Domain for more information about dropping a use case domain.

10.1.6 Using a Flexible Use Case Domain

This section explains how you can create, associate, disassociate, and drop a flexible use case domain.

> **Note:**
>
> You cannot alter a flexible use case domain, but as an alternative, you can disassociate the flexible domain from the tables, `DROP` the domain, recreate the domain, and re-associate it with the tables.

Topics:

- Creating a Flexible Use Case Domain
- Associating a Flexible Use Case Domain at Table Creation
- Associating a Flexible Domain with Existing Columns
- Disassociating a Flexible Use Case Domain from Columns
- Dropping a Flexible Use Case Domain

10.1.6.1 Creating a Flexible Use Case Domain

You can create a flexible use case domain that references other non-flexible domains (single and multi-column use case domains) and enables you to apply one of them to table columns depending on the context of the data. For example, you can create multi-column domains to validate address formats for each country. Table columns can only belong to one domain. So, to enable the database to use the address domain corresponding to each country for each row, create a flexible domain over the country domains. Associate columns with the flexible domain using the country as the discriminant column. Each row can then apply the address rules for the corresponding country domain.

Example 10-42 Creating a Flexible Use Case Domain for Temperature Readings

The following code creates a flexible use case domain called `temperature` from three domains, namely `celcius`, `fahrenheit`, and `kelvin`. The domains that are created for each temperature scale have appropriate absolute zero checks and display expressions. There is no `ELSE` clause in the flexible domain, so you can insert values for other temperature units, and temperature values for such temperature units are unconstrained.

```
CREATE DOMAIN celcius AS NUMBER
  CONSTRAINT abs_zero_c_c CHECK ( celcius >= -273.15 )
  DISPLAY celcius || ' °C';

CREATE DOMAIN fahrenheit AS NUMBER
  CONSTRAINT abs_zero_f_c CHECK ( fahrenheit >= -459.67 )
  DISPLAY fahrenheit || ' °F';

CREATE DOMAIN kelvin AS NUMBER
  CONSTRAINT abs_zero_k_c CHECK ( kelvin >= 0 )
  DISPLAY kelvin || ' K';
```

The following code creates a flexible domain that selects which domain to use based on the temperature units.

```
CREATE FLEXIBLE DOMAIN temperature (
  temp
 ) CHOOSE DOMAIN USING ( units char(1) )
 FROM (
  CASE units
    WHEN 'C' THEN celcius ( temp )
    WHEN 'F' THEN fahrenheit ( temp )
    WHEN 'K' THEN kelvin ( temp )
  END);
```

Example 10-43 Creating a Flexible Use Case Domain for Addresses

The following code creates multi-column use case domains to represent United States and United Kingdom (British) addresses, and a default address domain for other countries.

```
/* US addresses */
CREATE DOMAIN us_address AS (
  line_1  AS VARCHAR2(255 CHAR) NOT NULL,
```

```
  town    AS VARCHAR2(255 CHAR) NOT NULL,
  state   AS VARCHAR2(255 CHAR) NOT NULL,
  zipcode AS VARCHAR2(10 CHAR) NOT NULL
 ) CONSTRAINT us_address_c check (
   REGEXP_LIKE ( zipcode, '^[0-9]{5}(-[0-9]{4}){0,1}$' ));

/* British addresses */
CREATE DOMAIN gb_address AS (
  street   AS VARCHAR2(255 CHAR) NOT NULL,
  locality AS VARCHAR2(255 CHAR),
  town     AS VARCHAR2(255 CHAR) NOT NULL,
  postcode AS VARCHAR2(10 CHAR) NOT NULL
 ) CONSTRAINT gb_postcode_c check (
   REGEXP_LIKE (
    postcode, '^[A-Z]{1,2}[0-9][A-Z]{0,1} [0-9][A-Z]{2}$' ));

/* Default address */
CREATE DOMAIN global_address AS (
  line_1   AS VARCHAR2(255) NOT NULL,
  line_2   AS VARCHAR2(255),
  line_3   AS VARCHAR2(255),
  line_4   AS VARCHAR2(255),
  postcode AS VARCHAR2(10));
```

The following code creates a flexible domain that selects which multi-column address domain to use based on the country code.

```
CREATE FLEXIBLE DOMAIN address (
  line_1, line_2, line_3, line_4,
  postal_code
 )
 CHOOSE DOMAIN USING ( country_code VARCHAR2(2 char) )
 FROM (
  CASE country_code
    WHEN 'GB' THEN gb_address ( line_1, line_2, line_3, postal_code )
    WHEN 'US' THEN us_address ( line_1, line_2, line_3, postal_code )
    ELSE global_address ( line_1, line_2, line_3, line_4, postal_code )
  END);
```

> **Note:**
>
> To create a flexible domain, you must have the `EXECUTE` privilege on each constituent domain.

> **See Also:**
>
> *Oracle Database SQL Language Reference* for the syntactic and semantic information about creating a use case domain: CREATE DOMAIN

10.1.6.2 Associating a Flexible Use Case Domain at Table Creation

You can use the `CREATE TABLE` DDL to associate a flexible use case domain with a set of columns that are newly created by the new table. To add a flexible domain to a set of columns, specify the list of columns to be associated with the domain (in the domain column order), followed with the list of columns to be used as the discriminant (in discriminant column order in the flexible domain).

Example 10-44 Associating the `temperature` Flexible Domain with New Table Columns

Create a `sensor_readings` table using the `temperature` flexible domain, while specifying the discriminant column with the `USING` keyword.

```
CREATE TABLE sensor_readings (
  sensor_id integer, reading_timestamp TIMESTAMP,
  temperature_reading NUMBER,
  temperature_units   CHAR(1 CHAR),
  DOMAIN temperature ( temperature_reading )
    USING ( temperature_units ));
```

Example 10-45 Associating the `address` Flexible Domain with New Table Columns

The following code creates a new table called `addresses` and associates its columns with the `address` flexible domain.

```
CREATE TABLE addresses (
  line_1 VARCHAR2(255) NOT NULL,
  line_2 VARCHAR2(255),
  line_3 VARCHAR2(255),
  line_4 VARCHAR2(255),
  country_code   VARCHAR2(2 CHAR) NOT NULL,
  postal_code    VARCHAR2(10 CHAR),
  DOMAIN address (
    line_1, line_2, line_3, line_4, postal_code)
    USING ( country_code ));
```

> **Note:**
>
> The `DOMAIN` and `USING` keywords are mandatory when associating flexible domains.

10.1.6.2.1 Using DML on Columns Associated with Flexible Domains

The following are some examples of DML statements on the newly created table columns that have associated flexible use case domains.

Example 10-46 Using DML Commands on Columns Associated with the `temperature` Flexible Domain

```
INSERT INTO sensor_readings
VALUES ( 1, timestamp'2023-06-08 12:00:00', 21.1, 'C' ),
```

```
        ( 1, timestamp'2023-06-08 12:05:00', 21.2, 'C' ),
        ( 1, timestamp'2023-06-08 12:10:00', 20.9, 'C' ),
        ( 2, timestamp'2023-06-08 12:00:00', 68.5, 'F' ),
        ( 2, timestamp'2023-06-08 12:05:00', 68.1, 'F' ),
        ( 2, timestamp'2023-06-08 12:10:00', 68.9, 'F' ),
        ( 3, timestamp'2023-06-08 12:00:00', 290.23, 'K' ),
        ( 3, timestamp'2023-06-08 12:05:00', 289.96, 'K' ),
        ( 3, timestamp'2023-06-08 12:10:00', 289.65, 'K' ),
        ( 4, timestamp'2023-06-08 12:00:00', 528.15, 'R' ),
        ( 4, timestamp'2023-06-08 12:05:00', 528.42, 'R' ),
        ( 4, timestamp'2023-06-08 12:10:00', 527.99, 'R' );

SELECT sensor_id, reading_timestamp,
  DOMAIN_DISPLAY ( temperature_reading, temperature_units ) temp
  FROM  sensor_readings;
```

> **See Also:**
>
> SQL Functions for Use Case Domains for details about domain functions, such as `DOMAIN_DISPLAY` and `DOMAIN_ORDER`.

The output is:

```
 SENSOR_ID READING_TIMESTAMP
TEMP
---------- ------------------------------
-----------------------------------------
         1 08-JUN-2023 12.00.00.000000000 21.1
°C
         1 08-JUN-2023 12.05.00.000000000 21.2
°C
         1 08-JUN-2023 12.10.00.000000000 20.9
°C
         2 08-JUN-2023 12.00.00.000000000 68.5
°F
         2 08-JUN-2023 12.05.00.000000000 68.1
°F
         2 08-JUN-2023 12.10.00.000000000 68.9
°F
         3 08-JUN-2023 12.00.00.000000000 290.23
K
         3 08-JUN-2023 12.05.00.000000000 289.96
K
         3 08-JUN-2023 12.10.00.000000000 289.65 K
         4 08-JUN-2023 12.00.00.000000000
<null>
         4 08-JUN-2023 12.05.00.000000000
<null>
         4 08-JUN-2023 12.10.00.000000000 <null>
```

Example 10-47 Out-of-bounds Constraint Errors

The following values are out-of-bounds constraint errors for their respective temperature scales.

```
INSERT INTO sensor_readings
  VALUES ( 1, timestamp'2023-06-08 12:15:00', -400, 'C' );

INSERT INTO sensor_readings
  VALUES ( 2, timestamp'2023-06-08 12:15:00', -999, 'F' );

INSERT INTO sensor_readings
  VALUES ( 3, timestamp'2023-06-08 12:15:00',   -1, 'K' );
```

Example 10-48 Using DML on Columns Associated with the `address` Flexible Domain

```
-- Great Britian
INSERT INTO addresses ( line_1, line_3, country_code, postal_code )
  VALUES ( '10 Big street', 'London', 'GB', 'N1 2LA' );

-- United States
INSERT INTO addresses ( line_1, line_2, line_3, country_code, postal_code )
  VALUES ( '10 another road', 'Las Vegas', 'NV', 'US', '87654-3210' );

-- Tuvalu
INSERT INTO addresses ( line_1, country_code )
  VALUES ( '10 Main street', 'TV' );

SELECT * FROM addresses;
```

The output is:

```
LINE_1             LINE_2       LINE_3    LINE_4    COUNTRY_CODE
POSTAL_CODE
------------------ ------------ --------- --------- ---------------
-----------
10 Big street      <null>       London    <null>    GB              N1
2LA
10 another road    Las Vegas    NV        <null>    US
87654-3210
10 Main street     <null>       <null>    <null>    TV              <null>
```

The following `INSERT` command returns an error because it tries to insert UK address with US zip code.

```
INSERT INTO addresses ( line_1, line_3, country_code, postal_code )
  VALUES ( '10 Big street', 'London', 'GB', '12345-6789' );


ORA-11534: check constraint (schema.SYS_C0010286) due to domain constraint
schema.SYS_DOMAIN_C00639 of domain schema.ADDRESS violated
```

The following `INSERT` command returns an error because it tries to insert US address without values for the state.

```
INSERT INTO addresses ( line_1, line_2, country_code, postal_code )
  VALUES ( '10 another road', 'Las Vegas', 'US', '87654-3210' );


ORA-11534: check constraint (schema.SYS_C0010289) due to domain constraint
schema.SYS_DOMAIN_C00636 of domain schema.ADDRESS violated
```

10.1.6.3 Associating a Flexible Domain with Existing Columns

You can use the `ALTER TABLE` DDL with the `MODIFY` or `ADD` clauses to associate a flexible use case domain with an existing column or a newly added column in an existing table.

Example 10-49

The following code creates a new table called `temp_sensor_readings`.

```
CREATE TABLE temp_sensor_readings (
  sensor_id integer, reading_timestamp TIMESTAMP,
  temperature_reading NUMBER,
  temperature_units   CHAR(1 CHAR));
```

The following code associates the `temperature` flexible domain with an existing column called `temperature_reading`.

```
ALTER TABLE temp_sensor_readings
  MODIFY (temperature_reading, temperature_units)
  ADD DOMAIN temperature;
```

Guidelines

- The `DOMAIN` keyword is mandatory when associating flexible domains.
- The `USING` keyword is mandatory for `ALTER TABLE .. ADD` statement.
- You cannot have the same column associated with multiple flexible domains, whether as a domain column or as a discriminant column.
- You cannot have the column associated with the same domain, but with a different column positioning.

10.1.6.4 Disassociating a Flexible Use Case Domain from Columns

You can use the `ALTER TABLE` DDL with the `DROP` clause to disassociate a flexible use case domain from a column.

Example 10-50

The following code drops the `temperature` domain from the `temp_sensor_readings` table.

```
ALTER TABLE temp_sensor_readings
  MODIFY (temperature_reading, temperature_units) DROP DOMAIN;
```

Guidelines

- The domain name is not required because the database knows which columns are associated with which domains and one column can only be associated with one domain.
- You cannot specify other options for `ALTER TABLE ..MODIFY` with `ALTER TABLE ..DROP DOMAIN`.

10.1.6.5 Dropping a Flexible Use Case Domain

To drop a flexible use case domain, use the same syntax as used for a single-column use case domain.

Guidelines

- To drop a domain that is referenced in a flexible domain, use `DROP DOMAIN` with the `FORCE` option. Doing this also drops the flexible domain.
- To drop a flexible domain in the `FORCE` mode, you must have privileges to drop the constituent flexible domains.

> **See Also:**
>
> - *Oracle Database SQL Language Reference* for the syntactic and semantic information about dropping a use case domain: DROP DOMAIN.
> - Dropping a Use Case Domain for more information about dropping a use case domain.

10.1.7 Using an Enumeration Use Case Domain

This section explains how you can create and use an enumeration use case domain (enumeration domain).

Topics:

- About Enumeration Type
- Enumeration Domains Overview
- Creating an Enumeration Domain
- Associating an Enumeration Domain at Table Creation
- Associating an Enumeration Domain with Existing Columns

10.1.7.1 About Enumeration Type

An Enumeration Type (also called ENUM or enumeration) is a data type that consists of an ordered set of values. As a language construct, ENUM can be used to define a fixed set of permitted literals (named values) of a data input field. Based on your application requirements, you can explicitly define the valid values in the column specification when creating a table. Some good examples of enumerations are days and months, or directions such as North, South, East, and West.

10.1.7.2 Enumeration Domains Overview

From release 23ai (version 23.4) onwards, Oracle Database supports the use of the enumeration type for a domain.

A domain that is created as an enumeration type is called an enumeration domain. As such, an enumeration domain defines a list of predefined values that can be stored in a column, such as a list of possible states for a customer order: open, pending, shipped, and delivered. After creating an enumeration domain, you can use the domain as the data type of a table column. Enumeration domains simplify adding enumeration type columns to tables in Oracle SQL.

The following are the general rules and guidelines for creating and using enumeration domains.

- An enumeration domain contains a set of names, and optionally, a value corresponding to a name.
- The name in an enumeration must be a valid SQL identifier.
- The names are ordinary Oracle SQL identifiers, and therefore, must obey all the restrictions that are applicable to a valid identifier in Oracle SQL.
- Every specified value must be a literal.
- The value can be of any data type that is supported for a use case domain, but all the values used in the domain must be of the same data type.
- If the values are unspecified, the value of the first name is 1. The value for each subsequent name is one more than the previous name's value.
- You can associate many names with each value.
- The names can be double quoted to bypass the keyword restrictions, to make it case sensitive, or both.
- The names inside an enumeration domain can be used wherever a literal is allowed in a scalar SQL expression.
- An enumeration domain can be used as you would any other single-column domain. However, unlike a regular domain, an enumeration domain has a default check constraint and display expression.
- An enumeration domain can be used in the `FROM` clause of the `SELECT` statement, as if it were a table.

10.1.7.3 Creating an Enumeration Domain

To create an enumeration domain, use the `CREATE DOMAIN AS ENUM` command.

> **See Also:**
>
> *Oracle Database SQL Language Reference* for the syntactic and semantic information about creating a use case domain: CREATE DOMAIN

The following examples illustrate how you can create enumeration domains for various use cases.

Creating an Enumeration Domain for Order Status

The following code creates an `order_status` domain with only a set of names. The values for each name in the domain is as follows: New = 1, Open = 2, Shipped = 3, Closed = 4, and Canceled = 5.

```
CREATE DOMAIN order_status AS
 ENUM (
   New,
   Open,
   Shipped,
   Closed,
   Canceled
 );
```

Creating an Enumeration Domain with Double-quoted Names

Similar to ordinary identifiers, an enumeration name can be double quoted to bypass the keyword restrictions, to make it case sensitive, or both. The following example creates an enumeration domain called `CRUD` with the `CRUD` functions as names, but using the double quotes allows the names to bypass the keyword restrictions.

```
CREATE DOMAIN CRUD AS
 ENUM ( "Create",
        "Read",
        "Update",
        "Delete"
 );
```

Creating an Enumeration Domain for Status Codes

You can also create an enumeration domain with values assigned to names. The following domain called `Status_Code` has a status code value assigned to each state.

```
CREATE DOMAIN Status_Code AS
 ENUM (
   OK = 200,
   Not_Found = 404,
   Internal_Error = 500
 );
```

Creating an Enumeration Domain for the Days of a Week

The following code creates an enumeration domain called `Days_Of_Week` for all the days in a week. The value for each name in the domain is as follows: 0 for Sunday and Su, 1 for Monday and Mo, 2 for Tuesday and Tu, 3 for Wednesday and We, 4 for Thursday and Th, 5 for Friday and Fr, and 6 for Saturday and Sa. Therefore, each value is associated with two names.

```
CREATE DOMAIN Days_Of_Week AS
 ENUM (
   Sunday = Su = 0,
   Monday = Mo,
   Tuesday = Tu,
   Wednesday = We,
```

```
   Thursday = Th,
   Friday = Fr,
   Saturday = Sa
 );
```

Creating an Enumeration Domain for Job Titles

The following code creates an enumeration domain for the likely job titles in an organization.

```
CREATE DOMAIN Job_Title AS
 ENUM (
   Clerk = 'CLERK',
   Salesclerk = Salesperson = 'SALESCLERK',
   Manager = 'MANAGER',
   Analyst = 'ANALYST',
   President = 'PRESIDENT'
 );
```

Creating an Enumeration Domain for US Holidays

A value can also be a literal or a constant expression (and so it can be evaluated as part of the `CREATE DOMAIN` DDL). The following example creates an enumeration domain for US holidays with evaluated expressions.

```
CREATE DOMAIN US_Holidays_2023 AS
 ENUM (
   "New Years Day" = to_date('Jan  2, 2023', 'Mon DD, YYYY'),
   "MLK Jr. Day" = to_date('Jan 16, 2023', 'Mon DD, YYYY'),
   "Presidents Day" = to_date('Feb 20, 2023', 'Mon DD, YYYY'),
   "Memorial Day" = to_date('May 29, 2023', 'Mon DD, YYYY'),
   "Juneteenth" = to_date('Jun 19, 2023', 'Mon DD, YYYY'),
   "Independence Day" = to_date('Jul  4, 2023', 'Mon DD, YYYY'),
   "Labor Day" = to_date('Sep  4, 2023', 'Mon DD, YYYY'),
   "Columbus Day" = to_date('Sep  4, 2023', 'Mon DD, YYYY'),
   "Veterans Day" = to_date('Nov 11, 2023', 'Mon DD, YYYY'),
   "Thanksgiving Day" = to_date('Nov 23, 2023', 'Mon DD, YYYY'),
   "Christmas Day" = to_date('Dec 25, 2023', 'Mon DD, YYYY')
 );
```

Creating an Enumeration Domain for a Healthcare Application

The following is an example of an enumeration domain that a Healthcare Application might use:

```
CREATE DOMAIN Blood_Group AS
 ENUM (
   A_Positive = A_Pos = 'A +',
   A_Negative = A_Neg = 'A -',
   B_Positive = B_Pos = 'B +',
   B_Negative = B_Neg = 'B -',
   AB_Positive = AB_Pos = 'AB +',
   AB_Negative = AB_Neg = 'AB -',
   O_Positive = O_Pos = 'O +',
```

```
  O_Negative = O_Neg = 'O -'
);
```

The following are a few more use cases for enumeration domains in a healthcare application.

- Imaging Modality: X-Ray, MRI, CT, Ultrasound.
- Prescription Frequency: Once Daily, Twice Daily, Thrice Daily, As Needed.
- Patient Condition: Stable, Critical, Recovering, Terminal.
- Allergy Type: Food, Drug, Environmental, Insect, Animal, Other.
- BMI Category: Underweight, Normal, Overweight, Obese.

10.1.7.4 Associating an Enumeration Domain at Table Creation

Similar to a single-column use case domain, you can use an enumeration domain as the data type of a table column at the time of table creation. The following example creates an `orders` table and associates the `order_status` enumeration domain (created earlier) with the `status` column.

```
CREATE TABLE orders(
 id NUMBER,
 cust VARCHAR2(100),
 status order_status
);
```

Describing the `orders` table shows that the `status` column is actually a numeric column with a single-column domain. The actual values are stored in the `status` column as numbers representing the order status.

```
DESCRIBE orders;
```

The output is:

```
Name             Null?          Type
- - - - - - - -  - - - - - - -  - - - - - - - - - - - - - - -
ID                              NUMBER
CUST                            VARCHAR2(100)
STATUS                          NUMBER SCOTT.ORDER_STATUS
```

10.1.7.4.1 Using DML on Columns Associated with Enumeration Domains

The following are some examples of DML statements issued on the table columns that are associated with enumeration domains.

Using the `INSERT` Command on the `orders` Table

To insert data into the `orders` table, construct each row using the appropriate order status.

```
INSERT INTO orders VALUES
  (1, 'Costco', order_status.open),
```

```
  (2, 'BMW', order_status.closed),
  (3, 'Nestle', order_status.open);
```

The output is:

```
3 rows created.
```

You can list these rows out, but to see the enumeration values using their corresponding enumeration names, you can use the standard mechanism provided by domains, which is the `DOMAIN_DISPLAY()` function:

```
SELECT id, DOMAIN_DISPLAY(status) status FROM orders;
```

The output is:

```
                ID  STATUS
- - - - - - - - -  - - - - - - - - - -
                 1  OPEN
                 2  CLOSED
                 3  OPEN
```

The following `SELECT` statement shows that the actual values stored in the `status` column are numbers.

```
SELECT id, status FROM orders;
```

The output is:

```
                ID   STATUS
- - - - - - - - -  - - - -
                 1        2
                 2        4
                 3        2
```

Using the `UPDATE` Command on the `orders` Table

As with inserts, enumeration names can also be used in other DML statements, such as the `UPDATE` statement.

The following example updates the rows in the `orders` table to change the status from Closed to Canceled and Open to Closed.

```
UPDATE orders
  SET status = CASE status
                 WHEN order_status.closed
                   THEN order_status.cancelled
                 WHEN order_status.open
                   THEN order_status.closed
               END
  WHERE status IN(order_status.closed, order_status.open);
```

The output is:

```
2 rows updated.
```

To verify that the updates were made as expected, use the `SELECT` command with the `DOMAIN_DISPLAY` function again.

```
SELECT id, DOMAIN_DISPLAY(status) status FROM orders;
```

The output is:

```
                ID  STATUS
- - - - - - - - -  - - - - - - - - - -
                 1  CLOSED
                 2  CANCELLED
                 3  CLOSED
```

Since the underlying data type of the `status` column is just a number, you can also directly update the status with any numeric value.

```
UPDATE orders SET status = 2 WHERE status = 5;
```

The output is:

```
1 row updated.
```

The domain check constraint verifies that it is a valid domain value:

```
UPDATE orders SET status = -7;
```

The output is:

```
ORA -11534: check constraint ... violated
```

Because enumeration names are just placeholders for literal values they can be used anywhere SQL allows literals:

```
SELECT 2*order_status.cancelled;
```

The output is:

```
2*ORDER_STATUS.CANCELLED
-  - - - - - - - - - - - -
                        10
```

Using the `SELECT` Command on an Enumeration Domain

Unlike a regular domain, an enumeration domain can be treated as a table and queried using a `SELECT` statement.

```
SELECT * FROM order_status;
```

The output is:

```
ENUM_NAME          ENUM_VALUE
- - - - - - -     - - - - - -
NEW                         1
OPEN                        2
SHIPPED                     3
CLOSED                      4
CANCELLED                   5
```

However, just like a regular domain, an enumeration domain cannot be the target of a DML statement.

```
UPDATE order_status SET value = 4;
```

The output is:

```
ORA -04044: procedure , function , ... not allowed here
```

10.1.7.5 Associating an Enumeration Domain with Existing Columns

Similar to a regular domain, you can use the `ALTER-TABLE-MODIFY` command to add an enumeration domain to a column of an existing table. The following code assumes that `emp` is an existing table with `job`, `deptno`, and `comm` columns.

```
ALTER TABLE emp
 MODIFY(job)
  ADD DOMAIN Job_Title;
```

You can continue using the standard SQL statements, as follows:

```
UPDATE emp
 SET job = 'MANAGER'
  WHERE deptno = 20;
```

Additionally, you can also use an enumeration explicitly in your SQL statements, as follows:

```
UPDATE emp
 SET job = Job_Title.Salesperson
  WHERE comm IS NOT NULL;
```

10.1.8 Specifying a Data Type for a Domain

A domain data type can be one of Oracle data types. However, a qualified domain name must not collide with the qualified user-defined data types, or with Oracle built-in types.

If a column is associated with a domain, and the column data type is not specified, then the domain data type is used for the associated column as the default data type. If the associated column already has a data type, the column's data type is used.

If the domain data type is defined as non-`STRICT`, the associated column's data type only needs to be compatible with the domain data type, meaning that their data type must be the same, but the length, precision, and scale can be different. For instance, for a non-strict domain, you can associate a domain of `VARCHAR2(10)` with any `VARCHAR2` column.

If the domain data type is defined as `STRICT`, the associated column's data type must be compatible with the domain data type and also match the length, precision, and scale of the domain data type.

Example 10-51 Associating Columns with Domain Data Type

The following example creates a `year_of_birth` domain and a `email_dom` domain and associates the domains with columns to show the compatibility of domain and column data types.

```
DROP DOMAIN IF EXISTS year_of_birth;

CREATE DOMAIN year_of_birth AS NUMBER(4)
    CONSTRAINT CHECK ( (TRUNC(year_of_birth) = year_of_birth) AND
(year_of_birth >= 1900) );


DROP DOMAIN IF EXISTS email_dom;

CREATE DOMAIN email_dom AS VARCHAR2(100)
    CONSTRAINT email_chk check (REGEXP_LIKE (email_dom, '^(\S+)\@(\S+)\.
(\S+)$'));

DROP TABLE IF EXISTS newcustomers PURGE;
CREATE TABLE newcustomers (
  cust_Id NUMBER,
  cust_year_of_birth NUMBER(6) DOMAIN year_of_birth,
  cust_hq_email VARCHAR2(50),
  cust_office_email VARCHAR2(100),
  cust_rep_email VARCHAR2(100));

DESC newcustomers;
```

The output is:

```
 Name                          Null?    Type
 ----------------------------- -------- ----------------------------
 CUST_ID                                NUMBER
 CUST_YEAR_OF_BIRTH                     NUMBER(6) SH.YEAR_OF_BIRTH
```

```
 CUST_HQ_EMAIL                          VARCHAR2(50)
 CUST_OFFICE_EMAIL                      VARCHAR2(100)
 CUST_REP_EMAIL                         VARCHAR2(200)
```

The `cust_year_of_birth` column is defined as an Oracle data type: `Number`, and also associated with the `year_of_birth` domain, so the column's data type is assigned to the column. The `cust_year_of_birth` column inherits all the properties defined in the `year_of_birth` domain, such as constraint, display, and ordering properties.

The following example creates a `ukcustomers` table with a column associated with the `year_of_birth` domain, but without the column's data type:

```
CREATE TABLE ukcustomers (
  cust_Id NUMBER,
  cust_year_of_birth DOMAIN year_of_birth);

DESC ukcustomers;
```

The output is:

```
 Name                           Null?    Type
 ------------------------------ -------- ----------------------------
 CUST_ID                                 NUMBER
 CUST_YEAR_OF_BIRTH                      NUMBER(4) SH.YEAR_OF_BIRTH
```

Here, the `cust_year_of_birth` column is assigned the domain's data type, which is `NUMBER(4)`.

In the following example, the `DOMAIN` keyword is omitted.

```
CREATE TABLE incustomers (
  cust_id NUMBER,
  cust_year_of_birth year_of_birth);

DESC incustomers;
```

The output is:

```
 Name                           Null?    Type
 ------------------------------ -------- ----------------------------
 CUST_ID                                 NUMBER
 CUST_YEAR_OF_BIRTH                      NUMBER(4) SH.YEAR_OF_BIRTH
```

In the column definition clause, the domain clause must either replace the data type clause, or immediately follow it.

If a domain column data type is not defined as `STRICT`, you can associate a domain to any column with the same data type, irrespective of the column length.

The following ALTER commands succeed because the domain and column data type has the same data type and the column lengths are not checked for non-STRICT domains.

```
ALTER TABLE newcustomers
  MODIFY cust_hq_email DOMAIN email_dom;

ALTER TABLE newcustomers
  MODIFY cust_office_email DOMAIN email_dom;

ALTER TABLE newcustomers
  MODIFY cust_rep_email DOMAIN email_dom;
```

If a domain column data type is defined as STRICT, the domain association works only when the column and the domain have the same data type and their lengths also match.

```
DROP DOMAIN IF EXISTS email_dom;

CREATE DOMAIN email_dom AS VARCHAR2(100) STRICT
  CONSTRAINT email_chk check (REGEXP_LIKE (email_dom, '^(\S+)\@(\S+)\.
(\S+)$'));
```

The following ALTER command succeeds.

```
ALTER TABLE newcustomers
  MODIFY cust_office_email DOMAIN email_dom;
```

The following ALTER commands fail because the column length and the domain length do not match.

```
ALTER TABLE newcustomers
  MODIFY cust_hq_email DOMAIN email_dom;

ALTER TABLE newcustomers
  MODIFY cust_rep_email DOMAIN email_dom;
```

Table Column Data Type to non-STRICT Domain Column Data Type Compatibility

Each of the following points lists the compatible types. You can associate any table column with a domain column that has a compatible type. The table and domain columns can have different lengths, precisions, and scales for non-STRICT domains.

- NUMBER, NUMER(p), NUMBER(p, s), NUMERIC, NUMERIC(p), NUMERIC(p, s), DECIMAL, DECIMAL(p), DEC, DEC(p), INTEGER, INT, SMALLINT, FLOAT, FLOAT(p), REAL, DOUBLE_PRECISION
- CHAR(L), CHAR(L CHAR), CHAR(L BYTE), CHARACTER(L CHAR), CHARACTER(L BYTE), CHARACTER(L)
- NCHAR(L), NATIONAL CHARACTER(L), NATIONAL CHAR (L)
- VARCHAR2(L), VARCHAR2(L CHAR), VARCHAR2(L BYTE), CHAR VARYING(L CHAR), CHAR VARYING(L BYTE), CHAR VARYING(L), CHARACTER VARYING(L CHAR), CHARACTER VARYING(L BYTE), CHARACTER VARYING(L)

- NVARCHAR2(L), NATIONAL CHAR VARYING (L), NATIONAL CHARACTER VARYING (L)
- TIMESTAMP, TIMESTAMP(p)
- TIMESTAMP WITH TIME ZONE, TIMESTAMP(p) WITH TIME ZONE
- TIMESTAMP WITH LOCAL TIME ZONE, TIMESTAMP(p) WITH LOCAL TIME ZONE
- INTERVAL YEAR TO MONTH, INTERVAL YEAR(p) TO MONTH
- INTERVAL DAY TO SECOND, INTERVAL DAY(p) TO SECOND, INTERVAL DAY TO SECOND(s), INTERVAL DAY(p) TO SECOND(s)
- ROWID, UROWID, UROWID(p)
- RAW(p), RAW(p)

Table Column Data Type to STRICT Domain Column Data Type Compatibility

Each of the following points lists the compatible types. You can associate any table column with a domain column that has a compatible type. The table and domain columns must have an exact match for length, precision, and scale for STRICT domains.

- NUMBER(*), NUMBER
- NUMBER(p, 0), NUMERIC(p), NUMERIC(p, 0), DECIMAL(p), DEC(p), provided the table column data type and the domain column data type have the same precision.
- NUMBER(p, s), NUMERIC(p, s), DECIMAL(p, s), DEC(p, s), provided the table column data type and the domain column data type have the same precision and scale.
- NUMBER(*,0), NUMERIC(*), NUMERIC(*,0), DEC(*), DEC(*,0), DECIMAL(*), DECIMAL(*,0), INTEGER, INT, SMALLINT
- FLOAT(63), REAL
- FLOAT, FLOAT(126), DOUBLE PRECISION
- CHAR(L CHAR), CHAR(L BYTE), CHARACTER (L), provided the size in bytes is the same for the column data type and domain column data type. For example, CHAR (4 BYTE) can be associated with a STRICT domain column of CHAR(1 CHAR) if 1 character takes 4 bytes.
- NCHAR(L), NATIONAL CHARACTER(L), NATIONAL CHAR (L), provided the size in bytes is the same for the column data type and domain column data type.
- VARCHAR2(L CHAR), VARCHAR2(L BYTE), CHARACTER VARYING (L), CHAR VARYING(L), provided the size in bytes is the same for the column data type and domain column data type.
- NVARCHAR2(L), NATIONAL CHAR VARYING (L), NATIONAL CHARACTER VARYING (L), provided the size in bytes is the same for the column data type and domain column data type.
- TIMESTAMP, TIMESTAMP(6)
- TIMESTAMP WITH TIME ZONE, TIMESTAMP(6) WITH TIME ZONE
- TIMESTAMP WITH LOCAL TIME ZONE, TIMESTAMP(6) WITH LOCAL TIME ZONE
- INTERVAL YEAR TO MONTH, INTERVAL YEAR(2) TO MONTH
- INTERVAL DAY TO SECOND, INTERVAL DAY(2) TO SECOND, INTERVAL DAY TO SECOND(6), INTERVAL DAY(2) TO SECOND(6)
- ROWID, UROWID(10)
- UROWID, UROWID(4000), ROWID(4000)

Table Column Data Type to Domain Column Data Type Compatibility

The following column data types are only compatible with an equivalent domain column data type. For example, a `DATE` column type can be associated only with a `DATE` domain column type.

- `BINARY_FLOAT`
- `BINARY_DOUBLE`
- `DATE`
- `BLOB`
- `CLOB`
- `NCLOB`
- `BFILE`
- `LONG`
- `LONG RAW`
- `JSON`

Rules of Associating Table Column and Use Case Domain Based on Data Type

For a domain's data type of `VARCHAR2(L [CHAR|BYTE])`, let L-bytes be the maximum length in bytes corresponding to L, given that the National Language Support (NLS) setting has the session-level length semantics value in `NLS_LENGTH_SEMANTICS` as `BYTE`.

The following rules apply when you associate a column with the domain:

- If the domain is defined as non-`STRICT`, the domain can be associated with columns of data type `VARCHAR2(x)` for any x-bytes. For non-`STRICT` domains, L and x can be different.
- If the domain is defined as `STRICT`, the domain can be associated with columns of data type `VARCHAR2(x)` for any x-bytes = L-bytes. For `STRICT` domains, L and x must be the same number of bytes, even after converting L|x `CHAR` to `BYTES`, if needed.

For instance, if a domain data type specification is `VARCHAR2 STRICT`, and if `MAX_STRING_SIZE` is `STANDARD`, then the domain can associate with columns of `VARCHAR2(L BYTE)` data type, where L = 4000. If the current NLS settings in the session are such that at most 2 bytes are needed to represent a character, then a column of `VARCHAR2(2000 CHAR)` data type can be associated with the domain. If `MAX_STRING_SEMANTICS` is changed to `EXTENDED`, then columns of data type: `VARCHAR2(L BYTE)` for L = 32767 or `VARCHAR2(16383 CHAR)` can be associated with the domain.

Similar rules apply to `NVARCHAR2(L)`, `CHAR(L [CHAR|BYTE])`, and `NCHAR(L)`.

10.1.9 Changing the Use Case Domain Properties

As your application evolves, column definitions may need to change, and with that the properties of the associated use case domains must also change. At the time of altering a domain, you can only alter the annotations, display expression, and order expression for domains. The following suggests ways to change the other domain properties.

You can use these examples as a means to get started, or to build upon and adapt them to your specific data and business requirements. The first example is a manual method and the next one is an online method using the `dbms_redefinition` package.

Example 10-52 Changing the Domain Properties Manually

Suppose that you want to change the string length of an email domain field from 100 to 200. To make that change, you must complete the following steps:

- Alter the associated column's string length to 200.
- Drop the current domain associated with the column.
- Create a new domain with the email string length as 200.
- Re-associate columns associated with the current domain with the new domain.

If the current domain definition and column association are as follows:

```
CREATE DOMAIN email_dom AS VARCHAR2(100)
  CONSTRAINT email_chk CHECK (regexp_like (email_dom, '^(\S+)\@(\S+)\.(\S+)$'));

CREATE TABLE t1 (
  id NUMBER,
  email DOMAIN email_dom
);
```

To view the list of columns associated with the domain:

```
SELECT table_name, column_name
  FROM user_tab_columns
  WHERE domain_name = 'EMAIL_DOM';

TABLE_NAME              COLUMN_NAME
----------------------- -------------------
T1                      EMAIL
```

Alter the t1 table to change the email column with string size as 200:

```
ALTER TABLE t1 modify email varchar2(200);
```

Drop the domain with the `FORCE PRESERVE` option:

```
DROP DOMAIN email_dom FORCE PRESERVE;
```

Then, create a new domain with the email string size as 200:

```
CREATE DOMAIN email_dom AS VARCHAR2(200)
  CONSTRAINT email_chk CHECK (regexp_like (email_dom, '^(\S+)\@(\S+)\.(\S+)$'));
```

And, re-associate the new domain with the email column:

```
ALTER TABLE t1
  MODIFY email DOMAIN email_dom;
```

Example 10-53 Changing the Domain Properties Using the online `dbms_redefinition` Package

You can use the Oracle online table reorganization package called `dbms_redefinition` to change permitted values, such as updating which currencies are supported. For instance, to change the supported currencies in a currency domain, the required steps are:

- Create a new domain.
- Migrate columns associated with the current domain to use the new domain.

The following example has only basic error handling. There can be instances where an error can occur after several tables are migrated, such as when some data violates the new constraint. A complete solution would need to account for these scenarios.

The following code creates a domain with a constraint that allows the following currencies: USD, GBP, and EUR.

```
CREATE DOMAIN currency AS (
  amount AS NUMBER,
  iso_currency_code AS CHAR ( 3 CHAR )
 ) CONSTRAINT supported_currencies_c
    CHECK ( iso_currency_code in ( 'USD', 'GBP', 'EUR' ) );
```

Suppose that the following tables have columns associated with the `currency` domain.

```
CREATE TABLE order_items (
  order_id   INTEGER, product_id INTEGER,
  total_paid NUMBER(10, 2), currency_code CHAR ( 3 CHAR ),
  DOMAIN currency ( total_paid, currency_code ),
  PRIMARY KEY ( order_id, product_id )
);

CREATE TABLE product_prices (
  product_id    INTEGER,
  unit_price    NUMBER(10, 2),
  currency_code CHAR( 3 char ),
  DOMAIN currency ( unit_price, currency_code ),
  PRIMARY KEY ( product_id, currency_code )
);
```

Use the `INSERT` command to store some values into the `product_prices` and `order_items` tables.

```
INSERT INTO product_prices
VALUES (1, 9.99, 'USD'),
       (1, 8.99, 'GBP'),
       (1, 8.99, 'EUR');

INSERT INTO order_items
VALUES (1, 1, 9.99, 'USD'),
       (2, 1, 8.99, 'GBP'),
       (3, 1, 8.99, 'EUR');

COMMIT;
```

Suppose that your business is expanding and you want to support more currencies. You cannot modify the constraints directly, so you need an alternative approach that includes:

- Creating a new domain.
- Altering the `product_prices` and `order_items` tables to drop the existing domain from the associated columns while preserving the constraints.
- Altering the tables to add the new domain.
- Removing the preserved constraints from the tables.

However, when done online, these are blocking DDL statements. Instead, you can use the `dbms_redefinition` package to modify the constraints online.

You must create a new domain with the constraint including the newly supported currencies, and associate it with temporary tables. The new domain replaces the original domain.

```
CREATE DOMAIN currency_d as (
  amount            AS NUMBER,
  iso_currency_code AS CHAR ( 3 CHAR )
 ) CONSTRAINT supported_currencies_d_c
  CHECK ( iso_currency_code in ( 'USD', 'GBP', 'EUR', 'CAD', 'MXP', 'INR',
'JPY' ) );

CREATE TABLE order_items_tmp (
  order_id   INTEGER, product_id INTEGER,
  total_paid NUMBER(10, 2), currency_code CHAR ( 3 CHAR ),
  DOMAIN currency_d ( total_paid, currency_code ) ,
  PRIMARY KEY ( order_id, product_id )
);

CREATE TABLE product_prices_tmp (
  product_id    INTEGER,
  unit_price    NUMBER(10, 2),
  currency_code CHAR (3 CHAR),
  DOMAIN currency_d ( unit_price, currency_code ),
  PRIMARY KEY ( product_id, currency_code )
);
```

To make the code more reusable, you can create a `redefine_table` procedure that calls the `dbms_redefinition` procedures to copy the properties of the temporary table columns to the current table columns, and then swap the current table columns to the new domain.

```
DECLARE

  PROCEDURE redefine_table ( current_table VARCHAR2, staging_table VARCHAR2 )
AS
    num_errors pls_integer;
  BEGIN
    DBMS_REDEFINITION.CAN_REDEF_TABLE(user, current_table);
    DBMS_REDEFINITION.START_REDEF_TABLE(user, current_table, staging_table);

    DBMS_REDEFINITION.copy_table_dependents(
      uname               => user,
      orig_table          => current_table,
      int_table           => staging_table,
      copy_constraints    => false,
      num_errors          => num_errors);

    IF num_errors > 0 THEN
      dbms_redefinition.abort_redef_table(user, current_table, staging_table);
      raise_application_error ( -20001, num_errors || ' errors copying
dependents from ' || current_table || ' to ' || staging_table );
    ELSE
      dbms_redefinition.finish_redef_table(user, current_table,
staging_table);
    END IF;
```

```
  END redefine_table;

BEGIN

  FOR tabs IN (
    SELECT distinct table_name from user_tab_cols
    WHERE  domain_name = 'CURRENCY'
  ) LOOP
    redefine_table(tabs.table_name, tabs.table_name || '_TMP');
  END LOOP;

END;
/
```

Use DML on the `product_prices` table to see if the new currencies are now supported.

```
-- New currencies now supported
INSERT INTO product_prices
  VALUES (1, 9.99, 'CAD');

-- Invalid currencies raise exception
INSERT INTO product_prices
  VALUES (1, 9.99, 'N/A');

SELECT * FROM product_prices;
```

The output is:

```
PRODUCT_ID UNIT_PRICE CUR
---------- ---------- ---
         1       9.99 USD
         1       8.99 GBP
         1       8.99 EUR
         1       9.99 CAD
```

Clean up the temporary objects and old domain: `currency`.

```
DROP TABLE order_items_tmp PURGE;
DROP TABLE product_prices_tmp PURGE;
DROP DOMAIN currency;
```

> **See Also:**
>
> DBMS_REDEFINITION in *PL/SQL Packages and Types Reference* Guide.

10.1.10 SQL Functions for Use Case Domains

Domain functions enable you to work with use case domains more efficiently.

You can use the following SQL functions with use case domains:

- `DOMAIN_DISPLAY` returns the domain display expression for the domain that the argument is associated with.
- `DOMAIN_NAME` returns the qualified domain name of the domain that the argument is associated with. Note that if there is a public synonym to the domain, it is returned; otherwise domain name is returned in `domain_owner.domain_name` format.
- `DOMAIN_ORDER` returns the domains order expression for the domain that the argument is associated with.
- `DOMAIN_CHECK(domain_name, value1, value2, ...)` applies constraint conditions of `domain_name` (not null or check constraint) to the `value` expression. It also checks the values for type compatibility. There can be many values; the number of value expressions must match the number of columns in the domain or the statement raises an error.
- `DOMAIN_CHECK_TYPE(domain_name, value1, value2, ...)` verifies whether the input expressions are compatible with the domain's data type. There can be many values; the number of value expressions must match the number of columns in the domain or the statement raises an error.

> **See Also:**
>
> Domain Functions in *Oracle Database SQL Language Reference* for more information about using SQL functions for use case domains

10.1.11 Viewing Domain Information

You can use the dictionary views to get information about the domains. Dictionary views can also help identify columns that have different properties, such as constraint and default expressions when compared to their associated domain.

10.1.11.1 Dictionary Views for Use Case Domains

Dictionary views: `[USER|DBA|ALL]_DOMAINS` and `[USER|DBA|ALL]_DOMAIN_COLS` represent domains and provide the following information about the domain columns. For flexible domains, the views also include the domain selector expression.

- Domain name
- Domain owner
- Display and ordering expression
- Default value
- Data type of the domain
- Collation

The following dictionary views are available for use case domains:

- `ALL_DOMAINS` describes the domains accessible to the current user.
- `DBA_DOMAINS` describes all domains in the database.
- `USER_DOMAINS` describes the domains owned by the current user.
- `ALL_DOMAIN_COLS` describes columns of the domains accessible to the current user.
- `DBA_DOMAIN_COLS` describes columns of all domains in the database.

- `USER_DOMAIN_COLS` describes columns of the domains owned by the current user.
- `ALL_DOMAIN_CONSTRAINTS` describes constraint definitions in the domains accessible to the current user.
- `DBA_DOMAIN_CONSTRAINTS` describes constraint definitions in all domains in the database.
- `USER_DOMAIN_CONSTRAINTS` describes constraint definitions in the domains owned by the current user.

> **See Also:**
>
> *Oracle Database Reference* for more information about the following views that are used for use case domains: ALL_DOMAINS, DBA_DOMAINS, USER_DOMAINS, ALL_DOMAIN_COLS, DBA_DOMAIN_COLS, USER_DOMAIN_COLS, ALL_DOMAIN_CONSTRAINTS, DBA_DOMAIN_CONSTRAINTS, USER_DOMAIN_CONSTRAINTS

10.1.12 Built-in Use Case Domains

Oracle Database provides some built-in use case domains that you can use directly on table columns, for example, `email_d`, `ssn_d`, and `credit_card_number_d`. Built-in use case domains exist in all PDBs. When a new PDB is added into a CDB, the built-in use case domains automatically get created and added in the PDB.

The following built-in domains are supported in Oracle Database. These are categorized as follows:

- Identifier Built-in Domains
- Tech Built-in Domains
- Numeric Built-in Domains
- Miscellaneous Built-in Domains

Table 10-1 Identifier Built-in Domains

#	Domain Name	Description	Example	Data Type	Annotation	Constraint	Domain Order	Domain Display
1	`city_d`	Urban area or municipality	"San Francisco", "Mumbai", "Tokyo"	VARCHAR2(100)	address	-	-	-
2	`country_code_d`	A standardized two-letter or three-letter code assigned to each country or territory, typically defined by international standards such as ISO 3166.	"USA", "IND", "AUS"	VARCHAR2(3)	address	-	-	-

Table 10-1 (Cont.) Identifier Built-in Domains

#	Domain Name	Description	Example	Data Type	Annotation	Constraint	Domain Order	Domain Display
3	`country_d`	A distinct territorial and sovereign entity recognized as an independent nation-state, characterized by its own government, laws, and international recognition.	"USA", "India", "Mexico"	VARCHAR2(100)	address	-	-	-

Table 10-1 (Cont.) Identifier Built-in Domains

#	Domain Name	Description	Example	Data Type	Annotation	Constraint	Domain Order	Domain Display
4	credit_card_number_d	A unique series of digits assigned to a credit card for identification and transaction purposes.	"1234567890123456" "12345678901234567" "123456789012345678" "1234567890123456789"	VARCHAR2(20)	-	REGEX - ^[0-9]{12,19}$	-	WHEN length(credit_card_number_d)=15 THEN SUBSTR(credit_card_number_d,1,4) || ' ' || SUBSTR(credit_card_number_d,5,6) || ' ' || SUBSTR(credit_ca

Table 10-1 (Cont.) Identifier Built-in Domains

#	Domain Name	Description	Example	Data Type	Annotation	Constraint	Domain Order	Domain Display
								rd_number_d,11,5) WHEN length(credit_card_number_d)=16 THEN SUBSTR(credit_card_number_d,1,4) || ' ' || SUBSTR(credit_card_number_d,5,4) || ' ' ||

Table 10-1 (Cont.) Identifier Built-in Domains

#	Domain Name	Description	Example	Data Type	Annotation	Constraint	Domain Order	Domain Display
								SUBSTR(credit_card_number_d,9,4) || ' ' || SUBSTR(credit_card_number_d,13,4) WHEN length(credit_card_number_d)=14 THEN SUBSTR(credit_card_number_d,1,4) || ' '

Table 10-1 (Cont.) Identifier Built-in Domains

#	Domain Name	Description	Example	Data Type	Annotation	Constraint	Domain Order	Domain Display

								`||`
								
								`SUBSTR(credit_card_number_d, 5,6) || ' ' ||`
								
								`SUBSTR(credit_card_number_d, 11,4)`

5	`date_of_birth_d`	The specific day, month, and year on which an individual was born, representing their birth date. It is a fundamental piece of personal information used for identification, age verification, and record-keeping purposes. The displayed format depends on the date format used by the session.	01-JAN-91, 02/28/2023, 2023/5 4	DATE	person_info	-	-	-
6	`district_d`	A defined area or region within a city or country, often characterized by distinct administrative, social, or geographical features.	"Manhattan", "Brooklyn"	VARCHAR2(100)	address	-	-	-
7	`floor_d`	The level or story within a building where a specific unit or residence is situated, often indicated in its address.	"1", "1A", "Ground", "First"	VARCHAR2(20)	address	-	-	-

Table 10-1 (Cont.) Identifier Built-in Domains

#	Domain Name	Description	Example	Data Type	Annotation	Constraint	Domain Order	Domain Display
8	house_number_d	The numerical identifier assigned to a building or residence within a street or locality, typically used for postal addressing and navigation purposes.	"1-A", "10", "C1-H123"	VARCHAR2(50)	address	-	-	-
9	license_plate_d	A unique combination of letters, numbers, or symbols displayed on a vehicle's license plate, serving as a unique identifier for that vehicle.	"ABC123", "1234 XYZ", "DEF-456", "789 GHI", "JKL 987", "MNO-654", "PQR1234", "STU 5678", "VWX-9012", "YZA 3456"	VARCHAR2(100)	person_info	-	-	-
10	marital_status_d	A demographic attribute indicating an individual's legal relationship status with respect to marriage. Common categories include "single," "married," "divorced," "widowed," or "separated", among others.	"single", "married", "widowed"	VARCHAR2(50)	person_info	-	-	-
11	national_id_d	A unique identifier issued by a government to its citizens or residents for identification and administrative purposes. National IDs are used to access government services, prove identity, and facilitate transactions such as voting, travel, and financial activities.	"123-45-6789", "AB123456C", "123456789012", "123.456.789-01"	VARCHAR2(50)	person_info	-	-	-

Table 10-1 (Cont.) Identifier Built-in Domains

#	Domain Name	Description	Example	Data Type	Annotation	Constraint	Domain Order	Domain Display
12	passport_d	A government-issued travel document that certifies the holder's identity and nationality, allowing them to travel internationally.	"AB123456","123456789","ABC123456","E1234567","AB1234567","123456789AB"	VARCHAR2(100)	person_info	-	-	-
13	payment_card_id_d	A unique identifier associated with a payment card, such as a credit card, debit card, or prepaid card. This identifier distinguishes one card from another within a financial institution's system and is used for authorization, processing, and tracking of transactions.	"4012 8888 8888 1881","6011 1111 1111 1117","3714 4963 5398 431","5555 5555 5555 4444","5105 1051 0510 5100"	VARCHAR2(200)	person_info, payment_info	-	-	-
14	phone_d	A numerical sequence used for telecommunications, allowing individuals to connect through voice calls or text messages.	"1-555-555-5555", "44-20-1234-5678", "61-2-1234-5678", "49-30-12345678"	VARCHAR2(20)	person_info	-	-	-

Table 10-1 (Cont.) Identifier Built-in Domains

#	Domain Name	Description	Example	Data Type	Annotation	Constraint	Domain Order	Domain Display
15	`phone_number_d`	A numerical sequence used for telecommunications, allowing individuals to connect through voice calls or text messages.	"+1234567890123456" "1234567890 12345" "+9876543210 98765" "98765432109 8765" "+123" "123"	VARCHAR2(17)	-	REGEX - ^[+]{0,1}[0-9]{1,16}$	-	-
16	`postal_code_d`	A numerical or alphanumeric code used by postal services to identify specific geographic areas for mail delivery and sorting.	"SW1A 1AA", "M5H 2N2", "2000", "10117", "75001", "100-0001", "22010-000", "100000", "110001", "2000"	VARCHAR2(20)	address	-	-	-
17	`province_d`	A territorial division within a country, typically possessing its own government and administrative authority, particularly in federal or decentralized systems.	"Alberta", "Ontario"	VARCHAR2(100)	address	-	-	-
18	`social_media_id_d`	A unique identifier associated with an individual or entity's account on a specific social media platform. It serves as a distinct label or reference point within the platform's system, allowing for identification and interaction with specific users or profiles. This domain is typically used in conjunction with `social_media_type_d` domain that represents the social media platform.	"user123", "nyname@email.com", "myaccount11", "203"	VARCHAR2(100)	person_info	-	-	-

Table 10-1 (Cont.) Identifier Built-in Domains

#	Domain Name	Description	Example	Data Type	Annotation	Constraint	Domain Order	Domain Display
19	`social_media_type_d`	Refers to the classification or category of a social media platform, such as Facebook, Twitter, Instagram, or LinkedIn. This domain is typically used in conjunction with `social_media_id_d` domain that represents the user identifier of a social media platform.	"facebook", "x", "instagram"	VARCHAR2(100)	-	-	-	-
20	`ssn_d`	A unique nine-digit identifier assigned by the Social Security Administration to individuals in the United States.	"123-45-6789", "987-65-4321", "456-78-9012", "876-54-3210", "234-56-7890"	VARCHAR2(11)	-	REGEX - ^[0-9]{3}[-][0-9]{2}[-][0-9]{4}$	-	-
21	`state_d`	A territorial and administrative division within a country, often possessing its own government and legal system.	"California", "Texas"	VARCHAR2(100)	address	-	-	-
22	`street_d`	The name of a road or thoroughfare where a building or residence is located, forming part of its address.	"4th Main Road", "Purple Street 104"	VARCHAR2(200)	address	-	-	-
23	`us_license_plate_d`	A unique alphanumeric combination displayed on a vehicle's license plate in the United States, serving as a distinctive identifier for that vehicle.	"ABC-XY-1234", "DEF-GH-567", "JKL-MN-890", "OPQ-RS-4567", "TUV-WX-8901"	VARCHAR2(20)	person_info	REGEX - ^[A-Za-z]{1,3}-[A-Za-z]{1,2}-[0-9]{1,4}$	-	-

Table 10-1 (Cont.) Identifier Built-in Domains

#	Domain Name	Description	Example	Data Type	Annotation	Constraint	Domain Order	Domain Display
24	us_passport_d	A travel document issued by the United States government to its citizens for international travel.	"ABC123456", "XYZ789012", "DEF456789", "GHI012345", "12345678"	VARCHAR2(100)	person_info	-	-	-
25	us_phone_d	A series of digits used for telecommunication in the United States.	"123-456-7890", "1234567890", "(123) 456-7890", "(123)456-7890", "123.456.7890", "1234567890"	VARCHAR2(20)	person_info	REGEX - ^\D?(\d{3})\D?\D?(\d{3})\D?(\d{4})$	-	-
26	us_postal_code_d	United States postal code, typically referring to ZIP codes.	"10001-1234", "90210-5678", "60601-9876", "02110-5432", "33109-8765"	VARCHAR2(20)	address	REGEX - ^\d{5}([\-]\d{4})?$	-	-
27	us_ssn_d	A unique nine-digit identifier assigned by the Social Security Administration to individuals in the United States.	"123-45-6789", "123456789"	VARCHAR2(15)	person_info	-	-	-

Table 10-1 (Cont.) Identifier Built-in Domains

#	Domain Name	Description	Example	Data Type	Annotation	Constraint	Domain Order	Domain Display
28	`us_state_d`	Administrative region within the United States.	"CA", "NY", "TX", "FL", "WA", "OH", "PA", "IL", "GA", "MI", "VA", "NC", "NJ", "MD", "AZ", "CO", "MA", "TN", "IN", "MO"	VARCHAR2(2)	address	REGEX - ^(AE|AL|AK|AP|AS|AZ|AR|CA|CO|CT|DE|DC|FM|FL|GA|GU|HI|ID|IL|IN|IA|KS|KY|LA|ME|MH|MD|MA|MI|MN|MS|MO|MP|MT|NE|NV|NH|NJ|NM|NY|NC|ND|OH|OK|OR|PW|PA|PR|RI|SC|SD|TN|TX|UT|VT|VI|VA|WA|WV|WI|WY)$	-	-

Table 10-1 (Cont.) Identifier Built-in Domains

#	Domain Name	Description	Example	Data Type	Annotation	Constraint	Domain Order	Domain Display
29	us_vehicle_id_d	A unique identifier assigned to a vehicle in the United States, often referred to as a Vehicle Identification Number (VIN).	"1G1JC1441Y7167030", "5XYZ1234567890123", "WBAFA53541LM82357", "2G1WG5EK5B1160036", "3N1AB7AP2JY320999"	VARCHAR2(20)	person_info	REGEX - ^[A-HJ-NPR-Z0-9]{17}$	-	-
30	us_zip_d	An alphanumeric code used by the United States Postal Service (USPS) to facilitate mail delivery and address sorting.	"10001-1234", "90210-5678", "60601-9876", "02110-5432", "33109-8765"	VARCHAR2(15)	address	REGEX - ^\d{5}(-\d{4})?$	-	-
31	vehicle_id_d	A unique identifier assigned to a vehicle, typically used for registration, tracking, and identification purposes by government authorities and transportation agencies.	"UK11AP999"	VARCHAR2(100)	person_info	-	-	-

Table 10-1 (Cont.) Identifier Built-in Domains

#	Domain Name	Description	Example	Data Type	Annotation	Constraint	Domain Order	Domain Display
32	zip_d	A postal code used by postal services to efficiently route mail to specific geographic regions.	"SW1A 1AA", "M5H 2N2", "2000", "10117", "75001", "100-0001", "22010-000", "100000", "110001", "2000"	VARCHAR2(20)	address	-	-	-

Table 10-2 Tech Built-in Domains

#	Domain Name	Description	Example	Data Type	Annotation	Constraint	Domain Order	Domain Display
1	cidr_d	CIDR: Abbreviation for Classless Inter-Domain Routing, used for specifying the size of an IP network.	"192.168.1.0/24", "10.0.0.0/8"	VARCHAR2(18)	tech_info	REGEX - ^(([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])\.){3}([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])\/([1-9]|[1-2][0-9]|3[0-2])$	-	-

Table 10-2 (Cont.) Tech Built-in Domains

#	Domain Name	Description	Example	Data Type	Annotation	Constraint	Domain Order	Domain Display
2	`created_on_d`	Date and time when a file, record, or entity was created.	'1960-01-01 23:03:20'	TIMESTAMP	tech_info, timestamp	-	-	-
3	`description_d`	Textual information providing details or characteristics about something.	"Vintage red bicycle", "Cozy wooden cabin", "Sparkling crystal chandelier", "Rustic farmhouse kitchen", "Gentle ocean breeze"	VARCHAR2(4000)	-	-	-	-

Table 10-2 (Cont.) Tech Built-in Domains

#	Domain Name	Description	Example	Data Type	Annotation	Constraint	Domain Order	Domain Display
4	email_d	Electronic mail address used for digital communication.	"example@example.com", "user123@gmail.com"	VARCHAR2(4000)	person_info	REGEX - ^([a-zA-Z0-9!#$%&*+=?^_`{\|}~-]+(\.[A-Za-z0-9!#$%&*+=?^_`{|}~-]+)*)@((([a-zA-Z0-9]([a-zA-Z0-9-]*[a-zA-Z0-9])?\.)+[a-zA-Z0-9]([a-zA-Z0-9-]*[a-zA-Z0-9])?)$	-	-
5	encryption_function_d	Algorithm used to encode data for secure transmission or storage.	"AES", "DES", "RSA", "Blowfish", "Twofish", "RC4"	VARCHAR2(1000)	tech_info, encryption	-	-	-
6	encryption_value_d	Data transformed using encryption algorithms to conceal its meaning. This domain is typically used in conjunction with encryption_function_d domain that represents the encryption function that was used.	"5d41402abc4b2a76b9719d911017c592"	VARCHAR2(1000)	tech_info, encryption	-	-	-

Table 10-2 (Cont.) Tech Built-in Domains

#	Domain Name	Description	Example	Data Type	Annotation	Constraint	Domain Order	Domain Display
7	file_extension_d	Suffix added to the name of a computer file to denote its format or usage.	.txt", ".jpg", ".pdf", ".mp3"	VARCHAR2(10)	tech_info	-	-	-
8	file_size_d	Magnitude of data in a computer file as measured in an arbitrary unit, but typically measured in bytes.	100, 500, 1000, 2000, 5000	NUMBER	tech_info	CHECK >=0	-	-
9	hash_function_d	Algorithm used to map data of arbitrary size to fixed-size values.	"MD5", "SHA-1", "SHA-256", "SHA-512", "CRC32"	VARCHAR2(1000)	tech_info, hash	-	-	-
10	hash_value_d	Result of applying a hash function to a data set, typically used for data integrity verification. This domain is typically used in conjunction with hash_function_d domain that represents the hash function which was used.	"5d41402abc4b2a76b9719d911017c592"	VARCHAR2(1000)	tech_info, hash	-	-	-
11	hostname_d	Unique label assigned to a device within a network.	"example.com" "subdomain.example.com" "mail.example.com" "123example456.com"	VARCHAR2(255)	tech_info	REGEX - ^([a-zA-Z0-9]|[a-zA-Z0-9][a-zA-Z0-9\-]{0,61}[a-zA-Z0-9])(\.([a-zA-Z0-9]|[a-zA-Z0-9][a-zA-Z0-9\-]{0,61}[a-zA-Z0-9]))*$	-	-

Table 10-2 (Cont.) Tech Built-in Domains

#	Domain Name	Description	Example	Data Type	Annotation	Constraint	Domain Order	Domain Display
12	`id_number_d`	Unique identifier assigned to an individual, entity, or object. An equivalent domain: `id_string_d` can be used if the identifier is not numeric.	9876543210, 2468135790, 1357924680, 5555555555	NUMBER	-	-	-	-
13	`id_string_d`	Sequence of characters used to uniquely identify an entity. An equivalent domain: `id_number_d` can be used if the identifier is numeric.	"abc123DEF456"	VARCHAR2(4000)	-	-	-	-

Table 10-2 (Cont.) Tech Built-in Domains

#	Domain Name	Description	Example	Data Type	Annotation	Constraint	Domain Order	Domain Display
14	ip_address_d	Unique numerical label assigned to devices participating in a computer network.	"172.16.0.1", "255.255.255.0", "2001:0db8:85a3:0000:0000:8a2e:0370:7334"	VARCHAR2(46)	tech_info	REGEX - ^(([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])\.){3}([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])$ or ^([0-9a-fA-F]{4}[:]){7}[0-9a-fA-F]{4}$ or ^([0-9a-fA-F]{4}[:]){6}(([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])\.){3}([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])$	-	-

Table 10-2 (Cont.) Tech Built-in Domains

#	Domain Name	Description	Example	Data Type	Annotation	Constraint	Domain Order	Domain Display
15	ipv4_address_d	Numeric label assigned to devices participating in a computer network using the Internet Protocol version 4.	"192.168.0.1", "10.0.0.1", "172.16.0.1", "255.255.255.0", "127.0.0.1", "8.8.8.8", "172.31.255.255", "198.51.100.1"	VARCHAR2(15)	tech_info	REGEX - ^(([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])\.){3}([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])$	-	-
16	ipv6_address_d	Numerical label assigned to devices participating in a computer network using the Internet Protocol version 6.	"2001:0db8:85a3:0000:0000:8a2e:0370:7334"	VARCHAR2(46)	tech_info	REGEX - ^([0-9a-fA-F]{4}[:]){7}[0-9a-fA-F]{4}$ or ^([0-9a-fA-F]{4}[:]){6}(([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])\.){3}([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])$	-	-

Table 10-2 (Cont.) Tech Built-in Domains

#	Domain Name	Description	Example	Data Type	Annotation	Constraint	Domain Order	Domain Display
17	`last_modified_on_d`	Date and time when a file or resource was last modified.	'1960-01-01 23:03:20'	TIMESTAMP	tech_info, timestamp	-	-	-
18	`last_opened_on_d`	Date and time when data or information was last accessed.	'1960-01-01 23:03:20'	TIMESTAMP	tech_info, timestamp	-	-	-
19	`last_updated_on_d`	Date and time when a file or resource was last opened.	'1960-01-01 23:03:20'	TIMESTAMP	tech_info, timestamp	-	-	-
20	`mac_address_d`	Unique identifier assigned to a network interface controller for communications within a network.	"00-1A-2B-3C-4D" "FF-A1-B2-C3-D4" "ab-cd-ef-12-34" "00:1A:2B:3C:4D" "FF:A1:B2:C3:D4" "ab:cd:ef:12:34" "001A.2B3C.4D5E" "FFA1.B2C3.D4E5" "abcd.ef12.3456"	VARCHAR2(17)	tech_info	REGEX - ^([a-fA-F0-9]{2}[-]){5}[a-fA-F0-9]{2}$ or ^([a-fA-F0-9]{2}[:]){5}[a-fA-F0-9]{2}$ or ^([a-fA-F0-9]{4}[.]){2}[a-fA-F0-9]{4}$	-	-

Table 10-2 (Cont.) Tech Built-in Domains

#	Domain Name	Description	Example	Data Type	Annotation	Constraint	Domain Order	Domain Display
21	mime_type_d	Identifier for the type of data transmitted in an internet message.	"text/plain", "application/json", "image/jpeg", "application/pdf", "audio/mpeg", "video/mp4", "application/xml", "application/octet-stream", "application/vnd.ms-excel", "text/html"	VARCHAR2(100)	tech_info	-	-	-
22	sha1_d	Cryptographic hash function	"2fd4e1c67a2d28fced849ee1bb76e7391b93eb12"	CHAR(40)	tech_info, hash	REGEX - ^[0-9a-fA-F]{40}$	-	-
23	sha256_d	Cryptographic hash function	"5feceb66ffc86f38d952786c6d696c79c2dbc239dd4e91b46729d73a27fb57e9"	CHAR(64)	tech_info, hash	REGEX - ^[0-9a-fA-F]{64}$	-	-

Table 10-2 (Cont.) Tech Built-in Domains

#	Domain Name	Description	Example	Data Type	Annotation	Constraint	Domain Order	Domain Display
24	sha512_d	Cryptographic hash function	"cf83e1357eefb8bdf1542850d66d8007d620e4050b5715dc83f4a921d36ce9ce47d0d13c5d85f2b0ff8318d2877eec2f63b931bd47417a81a538327"	CHAR(128)	tech_info, hash	REGEX - ^[0-9a-fA-F]{128}$	-	-
25	short_name_d	Abbreviated or truncated form of a longer name or title.	"Ben", "Amy", "Max", "Mia", "Sam", "Zoe", "Leo", "Ava", "Eli", "Liv"	VARCHAR2(500)	-	-	-	-

Table 10-2 (Cont.) Tech Built-in Domains

#	Domain Name	Description	Example	Data Type	Annotation	Constraint	Domain Order	Domain Display
26	subnet_mask_d	Numeric value used in combination with an IP address to define the network and host portions.	"255.255.255.0", "255.255.0.0", "255.255.255.128", "255.0.0.0", "255.255.255.252"	VARCHAR2(15)	tech_info	REGEX - ^(((255\.){3}(255|254|252|248|240|224|192|128|0))|((255\.){2}(255|254|252|248|240|224|192|128|0)\.0)|((255\.)(255|254|252|248|240|224|192|128|0)(\.0+){2})|((255|254|252|248|240|224|192|128|0)(\.0){3}))$	-	-

Table 10-2 (Cont.) Tech Built-in Domains

#	Domain Name	Description	Example	Data Type	Annotation	Constraint	Domain Order	Domain Display
27	`uri_d`	Uniform Resource Identifier, string of characters used to identify a resource.	"https://www.example.com", "ftp://ftp.example.org", "mailto:user@example.net", "tel:+1234567890", "file:///path/to/file.txt"	VARCHAR2(4000)	tech_info	-	-	-
28	`uuid4_d`	Universally Unique Identifier (UUID) version 4, a randomly generated identifier.	"550e8400-e29b-41d4-a716-446655440000", "123e4567-e89b-12d3-a456-426655440000", "a0a0a0a0-b0b0-cccc-1234-567890abcdef"	CHAR(36)	tech_info	REGEX - ^[0-9a-fA-F]{8}([-][0-9a-fA-F]{4}){3}[-][0-9a-fA-F]{12}$	-	-

Table 10-2 (Cont.) Tech Built-in Domains

#	Domain Name	Description	Example	Data Type	Annotation	Constraint	Domain Order	Domain Display
29	website_d	Collection of related web pages accessible through the World Wide Web.	"www.example.com", "blog.example.org", "shop.example.net", "www.google.com", "www.facebook.com", "www.twitter.com", "www.github.com", "www.wikipedia.org", "www.nytimes.com", "www.amazon.com"	VARCHAR2(4000))	tech_info	-	-	-

Table 10-3 Numeric Built-in Domains

#	Domain Name	Description	Example	Data Type	Annotation	Constraint	Domain Order	Domain Display
1	count_d	Refers to the numerical tally or total quantity of items, occurrences, or entities within a set or group. It represents the number of elements present and is used to quantify and track the abundance, frequency, or extent of something.	5, 12, 100, 1000	NUMBER	-	-	-	-

Table 10-3 (Cont.) Numeric Built-in Domains

#	Domain Name	Description	Example	Data Type	Annotation	Constraint	Domain Order	Domain Display
2	`counter_d`	A metric type that represents a cumulative value that only ever increases over time. Counters are typically used to measure the total number of events or occurrences, such as the number of HTTP requests received, or the total bytes transferred.	1000, 50000, 8, 128, 56.4,	NUMBER	prometheus	-	-	-
3	`duration_d`	The length of time during which something continues or exists, measured from a starting point to an endpoint. Durations quantify the time elapsed between two events or within a specific period, providing a measure of time span, interval, or duration.	100, 1000, 250, 5000	NUMBER	-	-	-	-
4	`gauge_d`	Represents a single numerical value that can either increase or decrease over time, enabling monitoring of various metrics such as CPU usage, memory consumption, or network traffic.	1000, 50000, 8, 128, -22.78	NUMBER	prometheus	-	-	-
5	`histogram_count_d`	A metric associated with histograms. It represents the total count of observations or events recorded within the histogram's buckets over a specific time window. This count indicates how many times the observed metric falls within each bucket range, providing insight into the distribution of occurrences across different value ranges. This domain is typically used in conjunction with `histogram_sum_d`, `histogram_name_d`, and `histogram_upper_inclusive_bound_d` domains.	10, 100, 45, 123	NUMBER	prometheus, histogram	-	-	-

Table 10-3 (Cont.) Numeric Built-in Domains

#	Domain Name	Description	Example	Data Type	Annotation	Constraint	Domain Order	Domain Display
6	`histogram_name_d`	A metric type used to measure the distribution of observations or events over a specified range of values. Each histogram has a name, which is a unique identifier used to reference and query the metric data. This domain is typically used in conjunction with `histogram_sum_d`, `histogram_count_d`, and `histogram_upper_inclusive_bound_d` domains.	"data_sent", "data_received", "request_timed_out"	VARCHAR2(1000)	prometheus, histogram	-	-	-
7	`histogram_sum_d`	A metric associated with histograms. It represents the sum of all observed values within the histogram's buckets over a specific time window. This sum can provide insights into the total cumulative value of the observed metric, such as the total duration of HTTP requests or the total amount of resource consumption within a given time period. This domain is typically used in conjunction with `histogram_name_d`, `histogram_count_d`, and `histogram_upper_inclusive_bound_d` domains.	10, 100, 45, 123	NUMBER	prometheus, histogram	-	-	-
8	`histogram_upper_inclusive_bound_d`	The upper limit of a bucket range, where values falling within this range are considered to be included in that bucket. This boundary indicates the maximum value that can be included in the bucket. For example, if a bucket has an upper inclusive bound of 100, it means that values up to and including 100 are counted within that bucket. This domain is typically in conjunction with `histogram_sum_d`, `histogram_count_d`, and `histogram_name_d` domains.	100, 500, 10, -20	NUMBER	prometheus, histogram	-	-	-

Table 10-3 (Cont.) Numeric Built-in Domains

#	Domain Name	Description	Example	Data Type	Annotation	Constraint	Domain Order	Domain Display
9	histogram_upper_inclusive_bound_string_d	The textual representation of the upper inclusive bound of a bucket range. This string typically represents the maximum value that can be included in the bucket. For example, if a bucket has an upper inclusive bound string of "100", it means that values up to and including 100 are counted within that bucket.	"100", "500", "10", "-20"	VARCHAR2(1000)	prometheus, histogram	-	-	-
10	mean_d	The average value of a set of numbers, calculated by adding up all the values and dividing by the total count of numbers.	10, -2345, 4.67, 0	NUMBER	statistics	-	-	-
11	median_d	The middle value in a sorted list of numbers, separating the higher half from the lower half.	54, 20, 10, 1.45, -123	NUMBER	statistics	-	-	-
12	mode_d	The value that appears most frequently in a dataset.	4, 234, 100, -100, 0	NUMBER	statistics	-	-	-
13	negative_number_d	Denotes any numerical value less than zero. It represents quantities or values below the zero mark on the number line.	-1, -123, -1.456	NUMBER	-	CHECK <0	-	-
14	non_negative_number_d	Refers to any numerical value that is either zero or greater than zero. It includes zero and all positive numbers.	1, 123, 1.456, 0	NUMBER	-	CHECK >=0	-	-
15	non_positive_number_d	Refers to any numerical value that is either zero or less than zero. It includes both zero and negative numbers.	-1, -123, -1.456, 0	NUMBER	-	CHECK <=0	-	-
16	non_zero_number_d	Refers to any numerical value that is not equal to zero. It includes both positive and negative numbers, excluding zero	-1, -123, -1.456, 1, 123, 1.456	NUMBER	-	CHECK >0 or <0	-	-
17	percent_change_d	A measure that calculates the relative difference between two values expressed as a percentage, typically representing the change over a period of time. It represents a proportion out of 100 parts.	5, -5, 10, -10, -66.4	NUMBER	-	-	-	-

Table 10-3 (Cont.) Numeric Built-in Domains

#	Domain Name	Description	Example	Data Type	Annotation	Constraint	Domain Order	Domain Display
18	`percent_d`	A unit of measurement denoting a portion or fraction of a whole, expressed as a fraction of 100. It represents a proportion out of 100 parts and is commonly used to indicate relative quantities, rates, or comparisons, such as percentages of increase, decrease, or distribution. Percentages can be negative.	100, 98, 0.50, 75, -300	NUMBER	-	-	-	-
19	`percentile_d`	A statistical measure that indicates the percentage of data points in a distribution that are equal to or below a given value.	0, 10, 50 , 99, 33.33	NUMBER	-	CHECK >=0	-	-
20	`positive_number_d`	Refers to any numerical value greater than zero. It represents quantities, measurements, or values that are above the zero mark on the number line.	1, 123, 1.456	NUMBER	-	CHECK >0	-	-
21	`ratio_d`	A comparison of the magnitude of two quantities, often expressed as the quotient of one divided by the other.	0.5, 0.33, 1.5, 1	NUMBER	-	-	-	-
22	`standard_deviation_d`	The measure of the amount of variation or dispersion in a set of data.	0, 1, 10, 34, 56.56	NUMBER	statistics	CHECK >= 0	-	-
23	`unit_count_d`	The numerical quantity or tally representing the number of individual units of a particular item or entity. It indicates the quantity of units present and is used to measure the abundance or quantity of items within a set or group. This domain is typically used in conjunction with `unit_id_d` and `unit_price_d` that represent the ID and price (the monetary value in an arbitrary currency) of a unit respectively.	5, 10, 1, 0, 1000	NUMBER	unit	-	-	-

Table 10-3 (Cont.) Numeric Built-in Domains

#	Domain Name	Description	Example	Data Type	Annotation	Constraint	Domain Order	Domain Display
24	unit_id_d	A unique identifier or code assigned to a specific unit of a product, item, or entity. It distinguishes one unit from another and is used for tracking, inventory management, and reference purposes. This domain is typically used in conjunction with unit_count_d and unit_price_d that represent the count and price (the monetary value in an arbitrary currency) of a unit respectively.	50061, 50062, 123, 100001	NUMBER	unit	-	-	-
25	unit_price_d	The monetary value or cost associated with a single unit of a product, item, or entity. It represents the price charged or paid for each unit, and is used to calculate the total cost or revenue generated from the sale or purchase of units. This domain is typically used in conjunction with unit_count_d and unit_id_d that represent the count and ID of a unit respectively.	1, 15, 3.99, 500.67	NUMBER	unit	-	-	-
26	variance_d	The measure of the dispersion or spread of a set of data points around their mean.	0, 1, 10, 34, 56.56	NUMBER	statistics	CHECK >= 0	-	-

Table 10-4 Miscellaneous Built-in Domains

#	Domain Name	Description	Example	Data Type	Annotation	Constraint	Domain Order	Domain Display
1	binary_d	Base-2 numeral system representing numbers using only two symbols, typically 0 and 1. Each digit represents a power of 2. For example, the binary number 1010 represents the decimal number 10.	"01010001111", "0", "1", "00001111"	VARCHAR2(4000)	-	REGEX - ^[01]*$	-	-

Table 10-4 (Cont.) Miscellaneous Built-in Domains

#	Domain Name	Description	Example	Data Type	Annotation	Constraint	Domain Order	Domain Display
2	`currency_amount_d`	Refers to the numerical value assigned to a specific currency unit in a financial transaction or monetary exchange. It represents the quantity or amount of currency being transacted. This domain is typically used in conjunction with the `currency_code_d` domain that represents the currency in which the transaction is done.	10, 1000, 5000.67, 100000	NUMBER	currency	-	-	-
3	`currency_code_d`	Refers to a code, generally three lettered, that represents a specific currency in international transactions, such as USD for United States Dollar or EUR for Euro. This domain is typically used in conjunction with the `currency_amount_d` domain that represents the numerical value of the monetary transaction.	"USD", "INR", "EUR", "GBP"	VARCHAR2(3)	currency	-	-	-

Table 10-4 (Cont.) Miscellaneous Built-in Domains

#	Domain Name	Description	Example	Data Type	Annotation	Constraint	Domain Order	Domain Display
4	day_d	The name of the day of the week.	"SUNDAY", "MONDAY", ..., "SATURDAY"	CHAR(9)	-	List - ["SUNDAY", "MONDAY", "TUESDAY", "WEDNESDAY", "THURSDAY", "FRIDAY", "SATURDAY"]	Based on NLS Property 1. SUNDAY 2. MONDAY 3. TUEDAY 4. WEDDAY 5. THUDAY 6. FRIDAY 7. SATDAY OR 1. MONDAY 2. TUEDAY 3. WED	-

Table 10-4 (Cont.) Miscellaneous Built-in Domains

#	Domain Name	Description	Example	Data Type	Annotation	Constraint	Domain Order	Domain Display
							DAY 4. THUDAY 5. FRIDAY 6. SATDAY 7. SUNDAY	

Table 10-4 (Cont.) Miscellaneous Built-in Domains

#	Domain Name	Description	Example	Data Type	Annotation	Constraint	Domain Order	Domain Display
5	day_short_d	Abbreviated representation of the name of days of the week.	"SUN","MON", ..., "SAT"	CHAR(3)	-	List - ["SUN", "MON", "TUE", "WED", "THU", "FRI", "SAT"]	Based on NLS Property 1. SUN 2. MON 3. TUE 4. WED 5. THU 6. FRI 7. SAT OR 1. MON 2. TUE 3. WED 4. THU 5. FRI 6. SAT 7. SUN	-

Table 10-4 (Cont.) Miscellaneous Built-in Domains

#	Domain Name	Description	Example	Data Type	Annotation	Constraint	Domain Order	Domain Display
6	day_enum_d	Refers to an enumeration representing the seven days of the week. The domain supports long day names (MONDAY) and short day names (MON)	day_enum_d.MONDAY,day_enum_d.MON,day_enum_d.TUESDAY,day_enum_d.TUE	ENUM	calendar_info	ENUM - day_enum_d.MONDAY,day_enum_d.MON,day_enum_d.TUESDAY,day_enum_d.TUE	-	-
7	hexadecimal_d	Base-16 numeral system utilizing 16 symbols, including 0-9 and A-F to represent values from 10 to 15. Each digit represents a power of 16. For example, the hexadecimal number A8 represents the decimal number 168.	"001122AAFFF", "fffff", "00ffaa"	VARCHAR2(4000)	-	REGEX - ^[0-9a-fA-F]*$	-	-
8	latitude_d	Geographical coordinate that specifies the north-south position of a point on the Earth's surface relative to the equator. It is measured in degrees, ranging from 90 degrees north (at the North Pole) to 90 degrees south (at the South Pole).	37.773972	NUMBER	latlong, geographic	CHECK -90<=latitude<=90	-	-
9	longitude_d	Geographical coordinate that indicates the east-west position of a point on the Earth's surface relative to the Prime Meridian. It is measured in degrees, ranging from 180 degrees west to 180 degrees east.	-122.431297	NUMBER	latlong, geographic	CHECK -180<=longitude<=180	-	-
10	measure_d	Refers to the quantification or assessment of a particular attribute, characteristic, or quantity of an object, phenomenon, or system.	500, 1000, 1500, -22.3	NUMBER	-	-	-	-

Table 10-4 (Cont.) Miscellaneous Built-in Domains

#	Domain Name	Description	Example	Data Type	Annotation	Constraint	Domain Order	Domain Display
11	measure_type_d	This refers to the category or classification of a specific measurement. It defines the nature or type of the measurement being taken, such as weight, temperature, time, or quantity. This domain is used in conjunction with the measure_value_d domain that represents the numerical value of the measure.	“kg/cm^3”, “m”, “meters”, “celsius”	VARCHAR2(100)	measure	-	-	-
12	measure_value_d	The numerical or quantitative representation of a measurement within a given measure type. It indicates the magnitude or amount of the observed phenomenon. This domain is used in conjunction with the measure_type_d domain that represents the unit of the measure.	1000, 45.67, -300, 0	NUMBER	measure	-	-	-

Table 10-4 (Cont.) Miscellaneous Built-in Domains

#	Domain Name	Description	Example	Data Type	Annotation	Constraint	Domain Order	Domain Display
13	`month_d`	The name of the months.	"JANUARY", "FEBRUARY", ..., "DECEMBER"	CHAR(10)	-	List - ["JANUARY", "FEBRUARY", ..., "DECEMBER"]	1. JANUARY 2. FEBRUARY 3. MARCH 4. APRIL 5. MAY 6. JUNE 7. JULY 8. AUGUST 9. SEPTEMBER 10. OCTOBER 11. NOVEM	-

Table 10-4 (Cont.) Miscellaneous Built-in Domains

#	Domain Name	Description	Example	Data Type	Annotation	Constraint	Domain Order	Domain Display
							BER	
							12. DECEMBER	
14	month_short_d	The abbreviated representation of the name of months in a year.	"JAN", "FEB", ..., "DEC"	CHAR(3)	-	List - ["JAN", "FEB", ..., "DEC"]	1. JAN	-
							2. FEB	
							3. MAR	
							4. APR	
							5. MAY	
							6. JUN	
							7. JUL	
							8. AUG	
							9. SEP	
							10. OCT	
							11. NOV	
							12. DEC	

Table 10-4 (Cont.) Miscellaneous Built-in Domains

#	Domain Name	Description	Example	Data Type	Annotation	Constraint	Domain Order	Domain Display
15	month_enum_d	Refers to an enumeration that represents the twelve months of the year. The domain supports long month names (JANUARY) and short month names (JAN).	month_enum_d.JANUARY, month_enum_d.JAN, month_enum_d.FEBRUARY,month_enum_d.FEB	ENUM	calendar_info	ENUM - month_enum_d.JANUARY, month_enum_d.JAN, month_enum_d.FEBRUARY,month_enum_d.FEB, and so on.	-	-
16	octal_d	Base-8 numeral system utilizing eight symbols, usually 0 to 7. Each digit represents a power of 8. For instance, the octal number 12 represents the decimal number 10.	"12345", "7777", "01010"	VARCHAR2(4000)	-	REGEX - ^[0-7]*$	-	-
17	rank_d	The position given to something compared to others based on a particular standard, indicating its place in order. Rank is usually a positive integer.	1, 2,3, 10, 100, 5000	INTEGER	assessment	CHECK >= 1	-	-
18	rating_d	Denotes a numerical or qualitative assessment given to evaluate the quality, value, or performance of something based on predefined criteria. Ratings help users make informed decisions by providing an indication of satisfaction, effectiveness, or relevance.	-100, 0, 1, 45, 300	NUMBER	assessment	-	-	-
19	score_d	Represents a numerical evaluation or rating assigned to measure performance, quality, or success in various contexts. Scores provide a standardized way to assess and compare performance levels or outcomes across different situations.	-100, 0, 1, 45, 300	NUMBER	assessment	-	-	-

Table 10-4 (Cont.) Miscellaneous Built-in Domains

#	Domain Name	Description	Example	Data Type	Annotation	Constraint	Domain Order	Domain Display
20	time_zone_abbreviation_d	Shortened form or acronym representing the name of a specific time zone. It is typically composed of letters and is used to identify a region's time standard quickly. Examples include "EST" for Eastern Standard Time or "CET" for Central European Time. This domain can be used in the place of time_zone_utc_offset_d, which represents the timezone in numerical form.	"PST", "IST", "EST"	VARCHAR2(3)	timezone	-	-	-
21	time_zone_utc_offset_d	The numerical difference in hours and minutes between a specific time zone's local time and Coordinated Universal Time (UTC). It represents the deviation from the standard UTC time, which serves as a reference point for coordinating time globally. Positive offsets indicate locations ahead of UTC, while negative offsets indicate locations behind UTC. This domain can be used in the place of time_zone_abbreviation_d, which represents the acronym of the timezone.	"UTC+04:00", "UTC-03:30"	VARCHAR2(20)	timezone	-	-	-
22	year_d	The year in numeric form. The year 1 BCE is numbered 0, the year 2 BCE is numbered −1, and in general the year n BCE is numbered "−(n − 1)" (a negative number equal to 1 − n). This system of numbering is often referred to as "Astronomical Year Numbering."	-22, 1996, 2000, 1800	NUMBER(4)	-	-	-	-

10.2 Schema Annotations

This section explains how you can use schema annotations (hereinafter "annotations") for database objects.

For many applications, it is important to maintain additional property metadata for database objects such as tables, views, table columns, indexes, and domains. Annotations enable your applications to store and retrieve additional user-specific metadata about database objects and

table columns. Applications can use such metadata to help render effective user interfaces and customize application logic.

Topics:

10.2.1 Overview of Annotations

What are annotations and where can you use them?

Annotations are a lightweight declarative facility for developers to centrally register usage properties for database schema objects. Annotations are stored in dictionary tables and are available to any application looking to standardize behavior across common data in related applications. Annotations are not interpreted by the database in any way. They are custom data properties for database metadata - included for table columns, tables, and indexes, that applications can use as additional property metadata to render user interfaces or customize application logic.

Being a mechanism to define and store application metadata centrally in the database, annotations enable you to share the metadata information across applications, modules and microservices. You can add annotations to schema objects when you create new objects (using the `CREATE` statements) or modify existing objects (using the `ALTER` statements).

Annotating the data model with metadata provides additional data integrity, consistency and data model documentation benefits. Your applications can store user-defined metadata for database objects and table columns that other applications or users can retrieve and use. Storing the metadata along with the data guarantees consistency and universal accessibility to any user or application that uses the data.

An individual annotation has a name and an optional value. The name and the optional value are freeform text fields. For example, you can have an annotation with a name and value pair, such as `Display_Label 'Employee Salary'`, or you can have a standalone annotation with only a name, such as `UI_Hidden`, which does not need a value because the name is self-explanatory.

The following are further details about annotations.

- When an annotation name is specified for a schema object using the `CREATE` DDL statement, an annotation is automatically created.
- Annotations are additive, meaning that you can specify multiple annotations for the same schema object.
- You can add multiple annotations at once to a schema object using a single DDL statement. Similarly, a single DDL statement can drop multiple annotations from a schema object.
- An annotation is represented as a subordinate element to the database object to which the annotation is added. Annotations do not create a new object type inside the database.
- You can add annotations to any schema object that supports annotations provided you own or have alter privileges on the schema object. You do not need to schema qualify the annotation name.

- You can issue SQL queries on dictionary views to obtain all annotations, including their names and values, and their usage with schema objects.

10.2.2 Annotations and Comments

You can also annotate database objects such as tables and table columns using the `COMMENT` command. Comments that are associated with schema objects are stored in data dictionaries along with the metadata of the objects.

Annotations are simpler to use and have a broader scope than comments. Following are the major differences between comments and annotations:

- Comments are a commenting mechanism used to add metadata to only certain schema objects, such as tables and columns. Comments are not available for other schema objects, such as indexes, procedures, triggers, and domains.
- Comments do not have a name, they have only a freeform value.
- Comments are not additive meaning that you cannot add multiple comments for the same object. Specifying a new comment overwrites the prior comment for the corresponding table or column.
- You need separate DDL statements for comments whereas you can combine multiple annotations into one DDL statement.
- A separate set of dictionary views exists for different entities. For instance, there is a view for table comments and another view for column comments. Annotations, on the other hand, are unified across all object types, which makes them simpler to query and use.

> **See Also:**
>
> Comments for more information about using `COMMENTS`.

10.2.3 Supported Database Objects

Annotations are supported for the following database objects.

- Tables and table columns
- Views and view columns
- Materialized views and materialized view columns
- Indexes
- Domains and multi-column domain columns

10.2.4 Privileges Required for Using Annotations

To add or drop annotations, you require the `CREATE` or `ALTER` privilege on the schema object for which the annotation is specified in the `CREATE` or `ALTER` DDL statements.

You cannot create or remove annotations explicitly. Annotations are automatically created when they are used for the first time. Since annotations are stored as subordinate elements within the database object on which they are defined, adding annotations does not create a new object type inside the database.

10.2.5 DDL Statements for Annotations

This section explains the annotation syntax and provides the DDL statements to define or alter annotations for tables, table columns, views, materialized views, indexes, and domains.

Topics:

10.2.5.1 Annotation Syntax

The following snippet illustrates the annotation syntax that is used in DDL statements to define annotations for tables, table columns, views, materialized views, indexes, and domains.

```
annotations
         ::= 'ANNOTATIONS' '(' annotations_list ')'

annotations_list
         ::= ( 'ADD' ('IF NOT EXISTS' | 'OR REPLACE' )? | 'DROP' 'IF EXISTS'? | REPLACE)?
 annotation ( ',' ( 'ADD' ('IF NOT EXISTS' | 'OR REPLACE' )? | 'DROP' 'IF EXISTS'? |
REPLACE)?
 annotation )*

annotation
         ::= annotation_name annotation_value?

annotation_name
         ::= identifier

annotation_value
         ::= character_string_literal
```

<annotation_value> is a character string literal that can hold up to 4000 characters.

<annotation_name> is an identifier and has the following requirements:

- An identifier can have up to 1024 characters.
- If an identifier is a reserved word, you must provide the annotation name in double quotes.
- An identifier with double quotes can contain white characters.
- An identifier that contains only white characters is not allowed.

10.2.5.2 DDL Statements to Annotate a Table

You can use the following DDL statements to annotate a table when creating or altering the table.

Using the `CREATE TABLE` Statement

The following are examples of the `CREATE TABLE` statement with annotations.

The following example adds an annotation: `Operation` with a JSON value, and another annotation: `Hidden`, which is standalone and has no value.

```
CREATE TABLE Table1 (
  T NUMBER)
  ANNOTATIONS(Operations '["Sort", "Group"]', Hidden);
```

Adding an annotation can be preceded by the `ADD` keyword. The `ADD` keyword is considered to be the default operation, if nothing is specified.

The following example uses the optional `ADD` keyword to add the `Hidden` annotation (which is also standalone) to `Table2`.

```
CREATE TABLE Table2 (
  T NUMBER)
  ANNOTATIONS (ADD Hidden);
```

The `ADD` keyword is an implicit operation when annotations are defined, and can be omitted.

Using the `ALTER TABLE` Statement

In the following example, the `ALTER TABLE` command drops all annotation values for the following annotation names: `Operations` and `Hidden`.

```
ALTER TABLE Table1
  ANNOTATIONS(DROP Operations, DROP Hidden);
```

The following example has the `ALTER TABLE` command to add `JoinOperations` annotation with `Join` value, and to drop the annotation name: `Hidden`. When dropping an annotation, you need to only include the annotation name with the `DROP` command.

```
ALTER TABLE Table1
  ANNOTATIONS(ADD JoinOperations 'Join', DROP Hidden);
```

Multiple `ADD` and `DROP` keywords can be specified in one DDL statement.

Trying to re-add an annotation with a different value for the same object (for object-level annotations) or for the same column (for column-level annotations) raises an error. For instance, the following statement fails:

```
ALTER TABLE Table1
  ANNOTATIONS(ADD JoinOperations 'Join Ops');
```

The output is:

```
ORA-11552: Annotation name 'JOINOPERATIONS' already exists.
```

As an alternative, the annotation syntax allows you to replace an annotation value using the `REPLACE` keyword. The following statement replaces the value of `JoinOperations` with '`Join Ops`':

```
ALTER TABLE Table1
  ANNOTATIONS(REPLACE JoinOperations 'Join Ops');
```

The output is

```
Table altered.
```

Alternatively, to avoid an error when an annotation already exists, you can use the `IF NOT EXISTS` clause. The following statement adds the `JoinOperations` annotation only if it does not exist. If the annotation exists, the annotation value is unchanged and no error is raised.

```
ALTER TABLE Table1
  ANNOTATIONS(ADD IF NOT EXISTS JoinOperations 'Join Ops');
```

Similarly, dropping a non-existent annotation raises an error:

```
ALTER TABLE Table1 ANNOTATIONS(DROP Title);
```

The output is:

```
ORA-11553: Annotation name 'TITLE' does not exist.
```

To avoid the error, the `IF EXISTS` clause can be used, as follows:

```
ALTER TABLE Table1
  ANNOTATIONS(DROP IF EXISTS Title);
```

The output is:

```
Table altered.
```

> **See Also:**
>
> CREATE TABLE and ALTER TABLE in SQL Language Reference for complete clause changes and definitions.

10.2.5.3 DDL Statements to Annotate a Table Column

You can use the following DDL statements to annotate a table column when creating or altering the table.

Using the `CREATE TABLE` Statement

To add column-level annotations using a `CREATE TABLE` statement, specify the annotations as a part of the `column_definition` clause. Annotations are specified at the end, after inline constraints.

The following examples specify annotations for columns at table creation.

```
CREATE TABLE Table1
  (T NUMBER ANNOTATIONS(Operations 'Sort', Hidden));

CREATE TABLE Table2
  (T NUMBER ANNOTATIONS (Hidden));
```

The following example specifies table-level and column-level annotations for the `Employee` table.

```
CREATE TABLE Employee (
  Id NUMBER(5) ANNOTATIONS(Identity, Display 'Employee ID', "Group" 'Emp_Info'),
  Ename VARCHAR2(50) ANNOTATIONS(Display 'Employee Name', "Group" 'Emp_Info'),
  Sal NUMBER ANNOTATIONS(Display 'Employee Salary', UI_Hidden)
)
 ANNOTATIONS (Display 'Employee Table');
```

The `Employee` table in the previous example has column-level and object-level annotations with `Display` annotations defined at the column level and object level. You can define a new annotation with the same name as long as it corresponds to a different object, column pair. For instance, you cannot define another annotation with the `Display` name for the `Employee` table but you can define a new annotation "`Group`" for the `Sal` column. The annotation "`Group`" needs to be double quoted because it is a reserved word.

Using the `ALTER TABLE` Statement

To add column-level annotations using an `ALTER TABLE` statement, specify the annotations as a part of the `modify_col_properties` clause. Annotations are specified at the end, after inline constraints.

The following example adds a new `Identity` annotation for the column T of `Table1` table.

```
ALTER TABLE Table1
  MODIFY T ANNOTATIONS(Identity 'ID');
```

The following example adds a `Label` annotation and drops the `Identity` annotation.

```
ALTER TABLE Table1
  MODIFY T ANNOTATIONS(ADD Label, DROP Identity);
```

> **See Also:**
>
> CREATE TABLE and ALTER TABLE in SQL Language Reference for complete clause changes and definitions.

10.2.5.4 DDL Statements to Annotate Views and Materialized Views

You can use the following DDL statements to annotate a view and a materialized view.

Views and materialized views support annotations at the view level and column level. Column-level annotations are only supported for table views as a part of the column alias definition clause.

Using the `CREATE VIEW` Statement

The following example shows the view-level and column-level annotations:

```
CREATE OR REPLACE VIEW HighWageEmp
(
  Id ANNOTATIONS(Identity, Display 'Employee ID', "Group" 'Emp_Info'),
  Ename ANNOTATIONS (Display 'Employee Name',  "Group" 'Emp_Info'),
  Sal  ANNOTATIONS (Display 'Emp Salary')
)
ANNOTATIONS (Title 'High Wage Employee View')
AS SELECT * FROM EMPLOYEE WHERE Sal > 100000;
```

Using the `CREATE MATERIALIZED VIEW` Statement

The following example adds annotation at the view level in the `Materialized View` statement:

```
CREATE MATERIALIZED VIEW MView1
  ANNOTATIONS (Title 'Tab1 MV1', ADD Snapshot)
  AS SELECT * FROM Table1;
```

The following example adds annotation at the view level and column level in the `Materialized View` statement:

```
CREATE MATERIALIZED VIEW MView1(
  T ANNOTATIONS (Hidden))
 ANNOTATIONS (Title 'Tab1 MV1', ADD Snapshot)
 AS SELECT * FROM Table1;
```

Using the `ALTER VIEW` Statement

To provide support for annotations, the `ALTER VIEW` statement has a new sub-clause added to it that allows altering annotations at the view level. Other sub-clauses that are supported in the `ALTER VIEW` statement are: modifying view constraints, enabling recompilation, and changing `EDITIONABLE` property.

Column-level annotations for views cannot be altered. Previously added column-level annotations can be dropped only by dropping the view and recreating a new one.

The following example is an `ALTER VIEW` statement that drops the `Title` annotation and adds the `Identity` annotation at the view level.

```
ALTER VIEW HighWageEmp
  ANNOTATIONS(DROP Title, ADD Identity);
```

Using the `ALTER MATERIALIZED VIEW` Statement

The `ALTER MATERIALIZED VIEW` statement has a new sub-clause that is added to alter annotations globally at the materialized view level. Column-level annotations can be dropped only by dropping and recreating the materialized view.

The following `ALTER MATERIALIZED VIEW` statement drops `Snapshot` annotation from `MView1`.

```
ALTER MATERIALIZED VIEW MView1
  ANNOTATIONS(DROP Snapshot);
```

> **See Also:**
>
> CREATE VIEW, ALTER VIEW, CREATE MATERIALIZED VIEW, and ALTER MATERIALIZED VIEW in SQL Language Reference for complete clause changes and definitions.

10.2.5.5 DDL Statements to Annotate Indexes

You can use the following DDL statements to annotate an index.

Using the `CREATE INDEX` Statement

The following are examples of creating index-level annotations.

```
CREATE TABLE DEPARTMENT(
  DEPT_ID NUMBER,
  UNIT NUMBER);

CREATE INDEX I_DEPT_ID ON DEPARTMENT(DEPT_ID)
  ANNOTATIONS(Display 'Department Index');

CREATE UNIQUE INDEX UI_UNIT ON DEPARTMENT(UNIT)
  ANNOTATIONS(Display 'Department Unique Index');
```

Using the `ALTER INDEX` Statement

The following is an example of altering an index-level annotation.

```
ALTER INDEX I_DEPT_ID
  ANNOTATIONS(DROP Display, ADD ColumnGroup 'Group1');
```

> **See Also:**
>
> CREATE INDEX and ALTER INDEX in SQL Language Reference for complete clause changes and definitions.

10.2.5.6 DDL Statements to Annotate Domains

You can use the following DDL statements to annotate a domain.

You can specify annotations for domains at the domain level or at the column level. Annotations defined on domains are inherited to objects that reference the domain. Annotations are allowed only for regular domains and not for flexible domains.

To specify domain-level annotations for single-column domains, the syntax with parentheses (used for multi-column domains) is required to distinguish between column-level and object-level annotations. Otherwise, the annotations for single column domains are considered column level.

Using the `CREATE DOMAIN` Statement

The following example creates domain annotations by specifying column-level annotations for a single-column domain.

```
CREATE DOMAIN dept_codes_1 AS NUMBER(3)
  CONSTRAINT dept_chk_1 CHECK (dept_codes_1 > 99 AND dept_codes_1 != 200)
  ANNOTATIONS (Title 'Column level annotation');
```

The following examples specify a domain-level annotation for a single-column domain. This requires the use of the domains syntax for multi-column domains (with parentheses for columns).

```
CREATE DOMAIN dept_codes_2 AS (
  code AS NUMBER(3)
  CONSTRAINT dept_chk_2 CHECK (code > 99 AND code != 200))
  ANNOTATIONS (Title 'Domain Annotation');

CREATE DOMAIN HourlyWages AS Number(10)
  DEFAULT ON NULL 15
  CONSTRAINT MinimalWage CHECK (HourlyWages > = 7 and HourlyWages <=1000) ENABLE
  DISPLAY TO_CHAR(HourlyWages, '$999.99')
```

```
  ORDER ( -1*HourlyWages )
  ANNOTATIONS (Title 'Column level annotation');
```

The following example creates a multi-column domain with annotations at the column and domain levels.

```
CREATE DOMAIN US_City AS (
    name  AS VARCHAR2(50) ANNOTATIONS (Address),
    state AS VARCHAR2(50) ANNOTATIONS (Address),
    zip AS NUMBER ANNOTATIONS (Address)
  )
  CONSTRAINT City_CK CHECK(state in ('CA','AZ','TX') and zip < 100000)
  DISPLAY name || ', ' || state || ' , ' || TO_CHAR(zip)
  ORDER state || ', ' || TO_CHAR(zip) || ', ' || name
  ANNOTATIONS (Title 'Domain Annotation');
```

Using the `ALTER DOMAIN` Statement

Object-level annotations can be modified with `ALTER` statements. Column-level annotations for domains cannot be altered. Previously added column-level annotations can be dropped only by dropping the domain and recreating a new one.

The following example alters domain-level annotation for the `dept_codes_2` domain.

```
ALTER DOMAIN dept_codes_2
  ANNOTATIONS(DROP Title, ADD Name 'Domain');
```

> **See Also:**
>
> CREATE DOMAIN and ALTER DOMAIN in SQL Language Reference for complete clause changes and definitions.

10.2.5.7 Dictionary Table and Views

A dictionary table called `ANNOTATIONS_USAGE$` includes all the usage of annotations with schema objects, such as tables and views. When a new annotation name, value, or both are specified to a schema object, a new entry in `ANNOTATIONS_USAGE$` table is created. Similarly, when a schema object drops an annotation, the corresponding entry is dropped from the `ANNOTATIONS_USAGE$` table.

The following dictionary views track the list of annotations and their usage across all schema objects:

- `{DBA|USER|ALL|CDB}_ANNOTATIONS`
- `{DBA|USER|ALL|CDB}_ANNOTATIONS_USAGE`
- `{DBA|USER|ALL|CDB}_ANNOTATIONS_VALUES`

10.2.5.7.1 Querying Dictionary Views

You can run queries on dictionary views to obtain the list of annotations that are used for specific schema objects.

Here are examples of query statements issued on an 'EMP' table:

To obtain table-level annotations for the ‘EMP’ table:

```
SELECT * FROM USER_ANNOTATIONS_USAGE
  WHERE Object_Name = 'EMPLOYEE'
   AND Object_Type = 'TABLE'
   AND Column_Name IS NULL;
```

To obtain column-level annotations for the ‘EMP’ table:

```
SELECT * FROM USER_ANNOTATIONS_USAGE
  WHERE Object_Name = 'EMPLOYEE'
   AND Object_Type = 'TABLE'
   AND Column_Name IS NOT NULL;
```

To obtain column-level annotations for the ‘EMP’ table as a single JSON collection per column:

```
SELECT  U.Column_Name, JSON_ARRAYAGG(JSON_OBJECT(U.Annotation_Name, U.Annotation_Value))
  FROM USER_ANNOTATIONS_USAGE U
  WHERE Object_Name = 'EMPLOYEE' AND Object_Type = 'TABLE' AND Column_Name IS NOT NULL
  GROUP BY Column_Name;
```