8 Application Data Usage

This chapter explains what data use case domains and schema annotations are.

- Data Use Case Domains
  An **data use case domain** is a high-level dictionary object that belongs to a schema and encapsulates a set of optional properties and constraints.
- Schema Annotations
  For many applications, it is important to maintain additional property metadata for database objects such as tables, views, table columns, indexes, and domains.

Data Use Case Domains

An **data use case domain** is a high-level dictionary object that belongs to a schema and encapsulates a set of optional properties and constraints.

Databases typically have had limited understanding of how data is actually used by applications. Databases store data using primitive types such as `VARCHAR2`, `NUMBER` or `DATE`, while the knowledge that a stored value represents, for example, a credit card number or date of birth, exists only within application-level meta data or context. This approach increases individual application complexity, fragments usage awareness across tools and applications, and introduces the possibility of divergent semantics.

Past attempts to counter this divergence have been to add more usage-specific built-in data types to databases and provide support for user-defined types for extensibility. More elaborate database type systems have received limited adoption since they increase the mismatch with application language types, create non-portable application code, and increase development complexity since operations spanning different types are restricted.

Such restrictions on representing data usage can be avoided by using Oracle Database's *extended* domain concept as defined by the SQL Standard. Using this approach, a column can be declared both with a primitive data type such as `NUMBER`, as well as with a domain for data usage, such as "Temperature" or "Credit Score". Such a use case domain can optionally be associated with different usage properties such as check constraints, display properties, ordering rules, and others. Most importantly, centralized domain information can be used by applications in order to standardize operations without requiring application-level meta data; for example, to mask credit card numbers, to format phone numbers and currency values, to display percent values in a column as a pie-chart, and more.

Use case domains do not modify the underlying data type and can, therefore, also be added to existing data without breaking applications or creating portability issues. A use case domain can be declared for one or more columns in a table, and can also be used to represent variable usage scenarios in which the actual usage of data depends on data in other columns. Use case domains can be thought of as lightweight type modifiers that centrally document intended data usage for applications. Also, use case domains can be used to share annotations.

Dropping a table with an associated use case domain can produce different results than dropping a table with an associated PL/SQL type. The following rules apply when you drop a table that has an associated use case domain or PL/SQL type:

- If a table has been dropped without the `PURGE` clause, then it will still be available in the recycle bin (if the recycle bin is enabled) for potential future restoration.

- If the table has a column using a PL/SQL type or abstract data type, and another column associated to a use case domain, then:
  - Dropping the use case domain with the `FORCE` clause will still allow the table to be restored. The use case domain is simply disassociated from the column.
  - Dropping the PL/SQL type, with or without the `FORCE` clause, will purge the table from the recycle bin because it cannot be completely restored.

Characteristics of Use Case Domains:

- **Constraints** - domains can be used to share constraints across multiple tables and columns. These constraints may be disabled for deferrable constraints. Primary key and foreign key constraints are not permitted in domains.
- **Data types** - all built-in Oracle data types are permitted, except for long, long raw, and object types.
- **Default expression** - default expressions are allowed, which are the same as column defaults.
- **Case and accent insensitive searching** - domains allow for case insensitive searches which can be used in the `WHERE` and `ORDER BY` clauses.
- **Display expression** - display expressions allow for custom display rules. For example, displaying currencies in the local format.
- **Order expression** - an order expression allows for complex ordering. For example, ordering by normalized standard business currency such as US dollar amounts.

Use Case Domain Types:

- **Single column** - applies to a single data column.
- **Multi-column** - applies to multiple columns, such as currency.
- **Flexible domains** - based on other domains and must be compatible datatypes with the sub-domains. An example is address formats for different countries.
- **Enumeration** - consists of a set of names, and optionally, a value corresponding to a name.

Strict versus Non-Strict:

- **Non-strict** - a non-strict domain must match the given data type, but not necessarily the length or scale for numeric data.
- **Strict** - requires the data type, length, and scale to be an exact match. For example, currency codes which must be exactly two or three characters.

You can create an enumeration domain using Oracle SQL. Unlike regular domains, an enumeration domain has a default CHECK constraint as well as a display expression. The names inside an enumeration domain can be used wherever a literal is allowed in a scalar SQL expression, and the domain itself can be used in the FROM clause of a SELECT statement as if it were a table. An enumeration domain is a special type of usage domain and can be used just like any single column domain.

An enumeration domain consists of a set of names and, optionally, a value corresponding to a name. The name has to be a valid SQL identifier and every specified value must be a literal or, in the case of a multi-column enumeration, a list of literals. While the values can be of any data type that are supported for a data use case domain, all of them must be of the same data type.

For example, user SCOTT, who has been previously granted the `CREATE DOMAIN` privilege, creates a single-column enumerated domain for the job title.

```
CREATE DOMAIN Job_Title AS
  ENUM (
    Clerk                   = 'CLERK',
    Salesman = SalesPerson = 'SALESMAN',
    Manager                 = 'MANAGER',
    Analyst                 = 'ANALYST',
    President               = 'PRESIDENT'
  );
```

The domain `Job_Title` can now be used as the data type when creating the `EMP` table. For example:

```
CREATE TABLE emp
( empno     NUMBER(4)     CONSTRAINT emp_pk PRIMARY KEY,
  ename     VARCHAR2(10),
  job       Job_Title,
  mgr       NUMBER(4)     CONSTRAINT emp_mgr_fk REFERENCES emp(empno),
  hiredate  DATE,
  sal       NUMBER(7,2)
);
```

When inserting a new row of data, the enumerated domain may be used to specify the value for the `JOB` column. The following three INSERT statements are equivalent.

```
INSERT INTO emp VALUES (8000, 'PATEL', Job_Title.Salesman, 7698, TO_DATE('11-
MAR-2024', 'DD-MON-YYYY'), 3500);


INSERT INTO emp VALUES (8000, 'PATEL', Job_Title.SalesPerson, 7698,
TO_DATE('11-MAR-2024', 'DD-MON-YYYY'), 3500);


INSERT INTO emp VALUES (8000, 'PATEL', 'SALESMAN', 7698, TO_DATE('11-
MAR-2024', 'DD-MON-YYYY'), 3500);
```

Related Topics

- Using Application Usage Domains

Schema Annotations

For many applications, it is important to maintain additional property metadata for database objects such as tables, views, table columns, indexes, and domains.

While domains include built-in usage properties such as check constraints, collations, custom sort order, and others, for extensibility the Oracle database also provide the ability to add custom properties via the `ANNOTATIONS` mechanism for database metadata - including table columns, tables, indexes, and more. Applications often need to maintain additional property metadata especially for rendering user interfaces or customizing application logic.

Some examples of column level usage properties can include:

- Display Label: This may be different from the column name (for example, display the title "Salary" from a column named Employee_Salary)
- Column Group : In many cases, a column "group" is interesting for a user interface (for example, to group Street Number, Street Name, City, and Zip columns into an Address group)
- Format Mask: For example, a display mask of `$99,999.99` can be used by a user interface tool to render 56434 as `$56,434.00`
- Hide: Whether to show the column in the user interface to an end user (for example, don't show columns with sensitive information or system-added columns to a certain class of end users)
- Highlight: Whether the column should be displayed with a special highlight
- Allowed Operations: Allows the interface to determine whether to allow a column to support sorting, grouping, displaying a list of values, and others.

Table level usage properties can similarly be used to help applications record whether a table contains sensitive information, what its display name should be, which modules in the application it is owned or managed by, and more.

Most applications create their own repositories for this type of usage metadata, resulting in developer complexity and the possibility of divergence across applications, modules, and microservices.

Oracle Database schema annotations are a lightweight declarative facility for developers to centrally register usage properties for database schema objects. Annotations are stored directly inside the database in dictionary tables alongside the data model definition and the data itself, and available to any applications in order to standardize behavior across common data, but are not interpreted by the database in any way. They should be thought of as lightweight standardized markup for database metadata, for use by applications to register and process extended and custom usage properties.

Related Topics

- Oracle Database Database Development Guide