Bases de données
SQL

LDD
Types et domaines

SQL_02
v210a

2021-11-01

Christina.Khnaisser@USherbrooke.ca
Luc.Lavoie@USherbrooke.ca

© 2018-2021, **Μῆτις** (http://info.usherbrooke.ca/llavoie)
CC BY-NC-SA 4.0 (https://creativecommons.org/licenses/by-nc-sa/4.0/)

Plan

- Rappel et mise en garde
- DOMAIN (un survol)
  - Motivation
  - Syntaxe ISO
  - Spécificités PostgreSQL
  - Exemples
  - Transportabilité
  - Recommandations
- TYPE (un survol)
  - Motivation
  - Syntaxe ISO
  - Spécificités PostgreSQL
  - Exemples
  - Transportabilité
  - Recommandations

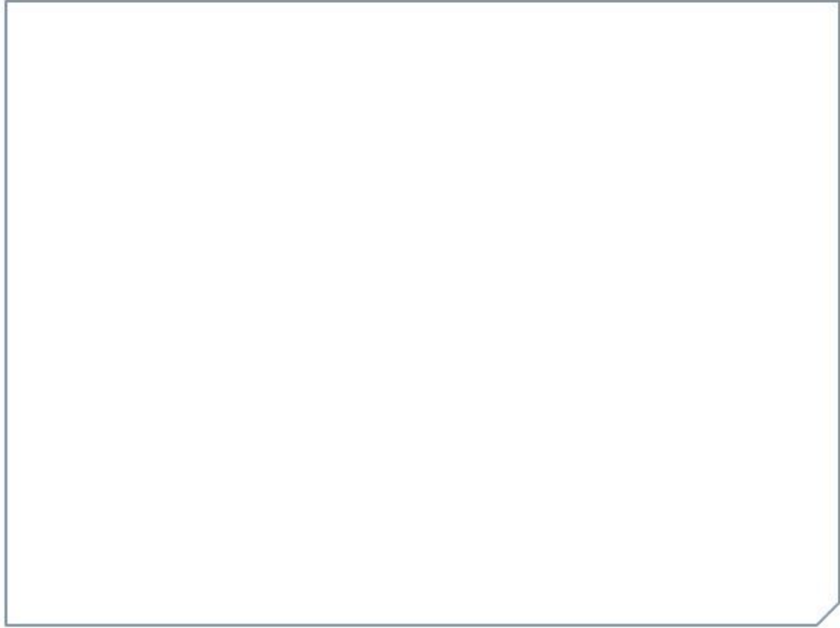
2021-11-01 SQL_02 : LDD Types et domaines (v210a) © 2018-2021, Μῆτις - CC BY-NC-SA 4.0
Département d'informatique, Faculté des sciences, Université de Sherbrooke, Québec

Rappel et mise en garde

- La théorie des types définit
  - **type de base** : la dénotation d'un ensemble fini de valeurs propres (non partagées avec un quelconque autre type de base);
  - **sous-type** : la dénotation d'un sous-ensemble d'une type de base défini par une contrainte.
- En SQL,
  - le constructeur TYPE permet de définir un **type de base** ;
  - le constructeur DOMAIN permet de définir un ***sous-type.***

2021-11-01 SQL_02 : LDD Types et domaines (v210a) © 2018-2021, Μῆτις - CC BY-NC-SA 4.0
Département d'informatique, Faculté des sciences, Université de Sherbrooke, Québec

Domaines



SQL_02 : LDD Types et domaines (v210a) © 2018-2021, Μῆτις - CC BY-NC-SA 4.0
Département d'informatique, Faculté des sciences, Université de Sherbrooke, Québec

DOMAIN
Motivation

- Assurer la cohérence d'un schéma en permettant la définition d'un type à partir d'un type de base et d'une contrainte.
- Faciliter la définition et l'évolution de schémas (particulièrement ceux de taille moyenne ou grande).

- Note
  - La représentation des valeurs du domaine est celle du type de base.

2021-11-01 SQL_02 : LDD Types et domaines (v210a) © 2018-2021, Mɏπɕ - CC BY-NC-SA 4.0
Département d'informatique, Faculté des sciences, Université de Sherbrooke, Québec

DOMAIN
Syntaxe ISO

définition de domaine ::=
CREATE DOMAIN *nom_de_domaine*
[AS] *type*
[DEFAULT *expression*]
[*contrainte_de_domaine* [...]]
[COLLATE collation]

contrainte de domaine ::=
[CONSTRAINT *nom_de_contrainte*]
CHECK (*condition*)
[<constraint_characteristics>]

2021-11-01 SQL_02 : LDD Types et domaines (v210a) © 2018-2021, Μῆτις - CC BY-NC-SA 4.0
Département d'informatique, Faculté des sciences, Université de Sherbrooke, Québec

<constraint characteristics> ::=
<constraint check time> [[NOT] DEFERRABLE] [<constraint enforcement>]
| [NOT] DEFERRABLE [<constraint check time>] [<constraint enforcement>]
| <constraint enforcement>

<constraint check time> ::=
INITIALLY DEFERRED | INITIALLY IMMEDIATE

<constraint enforcement> ::=
[NOT] ENFORCED

DOMAIN

Spécificités PosgreSQL

```
définition_de_domaine ::=
    CREATE DOMAIN nom_de_domaine
    [ AS ] type
    [ COLLATE collation ]
    [ DEFAULT expression ]
    [ contrainte_de_domaine [...] ]

contrainte_de_domaine ::=
    [ CONSTRAINT nom_de_contrainte ]
    { CHECK ( condition ) | [ NOT ] NULL }
```

2021-11-01 SQL_02 : LDD Types et domaines (v210a) © 2018-2021, Mηπς - CC BY-NC-SA 4.0
Département d'informatique, Faculté des sciences, Université de Sherbrooke, Québec

ref : https://docs.postgresql.fr/14/sql-createdomain.html

CREATE DOMAIN

CREATE DOMAIN — Définir un nouveau domaine

Synopsis

CREATE DOMAIN ***nom*** [AS] ***type_donnee*** [COLLATE ***collation***] [DEFAULT ***expression***] [***contrainte*** [...]] où ***contrainte*** est : [CONSTRAINT ***nom_contrainte***] { NOT NULL | NULL | CHECK (***expression***) }

Description

CREATE DOMAIN crée un nouveau domaine. Un domaine est essentiellement un type de données avec des contraintes optionnelles (restrictions sur l'ensemble de valeurs autorisées). L'utilisateur qui définit un domaine devient son propriétaire.

Si un nom de schéma est donné (par exemple, CREATE DOMAIN monschema.mondomaine ...), alors le domaine est créé dans le schéma spécifié. Sinon, il est créé dans le schéma courant. Le nom du domaine doit être unique parmi les types et domaines existant dans son schéma.

Les domaines permettent d'extraire des contraintes communes à plusieurs tables et de les regrouper en un seul emplacement, ce qui en facilite la maintenance. Par exemple, plusieurs tables pourraient contenir des colonnes d'adresses email, toutes nécessitant la même contrainte de vérification (CHECK) permettant de vérifier que

le contenu de la colonne est bien une adresse email. Définissez un domaine plutôt que de configurer la contrainte individuellement sur chaque table.

Pour pouvoir créer un domaine, vous devez avoir le droit USAGE sur le type sous-jacent.

Paramètres

nomLe nom du domaine à créer (éventuellement qualifié du nom du schéma).

type_donneesLe type de données sous-jacent au domaine. Il peut contenir des spécifications de tableau.

collationUn collationement optionnel pour le domaine. Si aucun collationnement n'est spécifié, le collationnement utilisé par défaut est celui du type de données. Le type doit être collationnable si COLLATE est spécifié.

DEFAULT ***expression***La clause DEFAULT permet de définir une valeur par défaut pour les colonnes d'un type de données du domaine. La valeur est une expression quelconque sans variable (les sous-requêtes ne sont pas autorisées). Le type de données de l'expression par défaut doit correspondre à celui du domaine. Si la valeur par défaut n'est pas indiquée, alors il s'agit de la valeur NULL.

L'expression par défaut est utilisée dans toute opération d'insertion qui ne spécifie pas de valeur pour cette colonne. Si une valeur par défaut est définie sur une colonne particulière, elle surcharge toute valeur par défaut du domaine. De même, la valeur par défaut surcharge toute valeur par défaut associée au type de données sous-jacent.

CONSTRAINT ***nom_contrainte***Un nom optionnel pour une contrainte. S'il n'est pas spécifié, le système en engendre un.

NOT NULLLes valeurs de ce domaine sont protégées comme les valeurs NULL. Cependant, voir les notes ci-dessous.

NULLLes valeurs de ce domaine peuvent être NULL. C'est la valeur par défaut.

Cette clause a pour seul but la compatibilité avec les bases de données SQL non standard. Son utilisation est découragée dans les applications nouvelles.

CHECK (***expression***)Les clauses CHECK spécifient des contraintes d'intégrité ou des tests que les valeurs du domaine doivent satisfaire. Chaque contrainte doit être une expression produisant un résultat booléen. VALUE est obligatoirement utilisé pour se référer à la valeur testée. Les expressions qui renvoient TRUE ou UNKNOWN réussissent. Si l'expression produit le résultat FALSE, une erreur est rapportée et la valeur n'est pas autorisée à être convertie dans le type du domaine.

Actuellement, les expressions CHECK ne peuvent ni contenir de sous-requêtes ni se référer à des variables autres que VALUE.

Quand un domaine dispose de plusieurs contraintes CHECK, elles seront testées dans l'ordre alphabétique de leur nom. (Les versions de PostgreSQLantérieures à la 9.5 n'utilisaient pas un ordre particulier pour la vérification des contraintes CHECK.)

Notes

Les contraintes de domaine, tout particulièrement NOT NULL, sont vérifiées lors de la conversion d'une valeur vers le type du domaine. Il est possible qu'une colonne du type du domaine soit lue comme un NULL bien qu'il y ait une contrainte spécifiant le contraire. Par exemple, ceci peut arriver dans une requête de jointure externe si la colonne de domaine est du côté de la jointure qui peut être NULL. En voici un exemple :

INSERT INTO tab (domcol) VALUES ((SELECT domcol FROM tab WHERE false));
Le sous-SELECT vide produira une valeur NULL qui est considéré du type du domaine, donc aucune vérification supplémentaire de la contrainte n'est effectuée, et l'insertion réussira.

Il est très difficile d'éviter de tels problèmes car l'hypothèse générale du SQL est qu'une valeur NULL est une valeur valide pour tout type de données. Une bonne pratique est donc de concevoir les contraintes du domaine pour qu'une valeur NULL soit acceptée, puis d'appliquer les contraintes NOT NULL aux colonnes du type du domaine quand cela est nécessaire, plutôt que de l'appliquer au type du domaine lui-même.

PostgreSQL suppose que les conditions des contraintes CHECK sont immuables, c'est-à-dire qu'elles produisent toujours les mêmes résultats pour les mêmes valeurs d'entrée. Cette supposition justifie que l'examen des contraintes CHECK est effectué seulement quand une valeur est initialement convertie vers le type domaine, et pas à d'autres moments. (C'est essentiellement le même traitement que les contraintes CHECK s'appliquant aux tables, comme décrit dans **Section 5.4.1**.)

Un exemple typique contrevenant à cette supposition consiste à faire référence à une fonction définie par l'utilisateur dans l'expression CHECK, puis de modifier le comportement de cette fonction. PostgreSQL n'interdit pas cela, mais il ne pourra pas remarquer qu'il y a des valeurs stockées dans le type du domaine qui seraient en violation de la contrainte CHECK. Cette situation peut ainsi provoquer l'échec du rechargement d'une sauvegarde faite par export. La méthode recommandée pour mener à bien ce type de changement consiste à supprimer la contrainte (en utilisant ALTER DOMAIN), à changer la définition de la fonction, puis à remettre la contrainte, ce qui la testera sur les données stockées.

Exemples

Créer le type de données code_postal_us, et l'utiliser dans la définition d'une table. Un test d'expression rationnelle est utilisé pour vérifier que la valeur ressemble à un code postal US valide :

CREATE DOMAIN code_postal_us AS TEXT CHECK(VALUE ~ '^\d{5}$' OR VALUE ~ '^\d{5}-\d{4}$'); CREATE TABLE courrier_us (id_adresse SERIAL PRIMARY KEY, rue1 TEXT NOT NULL, rue2 TEXT, rue3 TEXT, ville TEXT NOT NULL, code_postal code_postal_us NOT NULL);

Compatibilité

La commande CREATE DOMAIN est conforme au standard SQL.

Voir aussi

ALTER DOMAIN, DROP DOMAIN

DOMAIN
Exemples

```
CREATE  DOMAIN Cardinal
  INTEGER
  CHECK (VALUE >= 0);

CREATE DOMAIN Telephone
  VARCHAR(13)
  CHECK (VALUE SIMILAR TO '[0-9]{8,13}');

CREATE DOMAIN TauxEscompte
  NUMERIC(3,2)
  CHECK (VALUE BETWEEN 0.0 AND 1.00);
```

2021-11-01 SQL_02 : LDD Types et domaines (v210a) © 2018-2021, Μῆτις - CC BY-NC-SA 4.0
Département d'informatique, Faculté des sciences, Université de Sherbrooke, Québec

Faire la représentation graphique au tableau

DOMAIN
Transportabilité

- Disponibilité variable
- En cas d'absence, il faut alors utiliser
    CREATE TYPE

  dont
  - la syntaxe et l'usage varient d'un dialecte à l'autre;
  - la sémantique fait en sorte que les types ne peuvent être substitués simplement aux domaines;
  - dont la dénotation et les règles de compatibilité des valeurs diffèrent de celles des valeurs de domaine.

2021-11-01 SQL_02 : LDD Types et domaines (v210a) © 2018-2021, Mηπς - CC BY-NC-SA 4.0
Département d'informatique, Faculté des sciences, Université de Sherbrooke, Québec

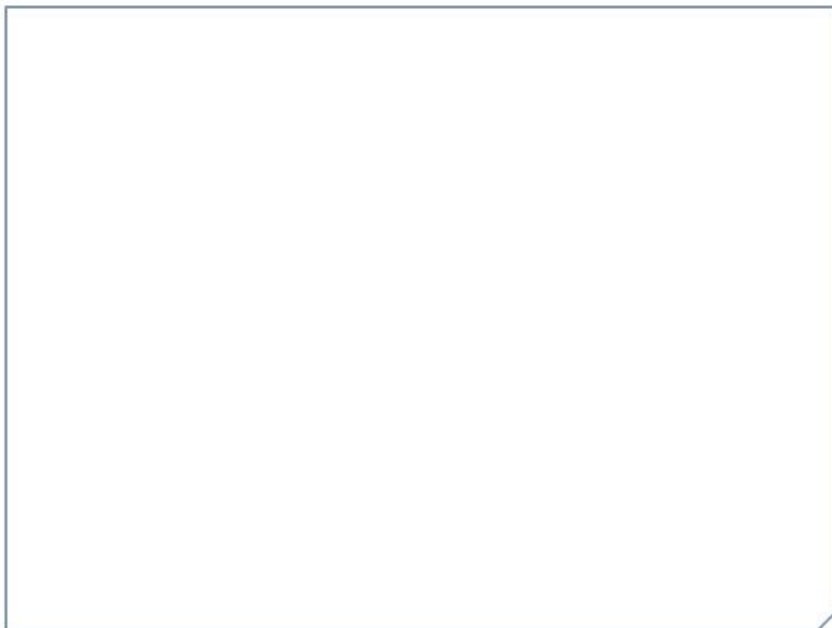
Faire la représentation graphique au tableau

DOMAIN
Recommandations

- Utiliser les DOMAIN si la pérennité d'utilisation du SGBD (e.a.: PostgreSQL) est bonne.
- Ne pas utiliser la contrainte NOT NULL par souci de transportabilité et d'homogénéité dans le traitement des types.

2021-11-01 SQL_02 : LDD Types et domaines (v210a) © 2018-2021, Mητις - CC BY-NC-SA 4.0
Département d'informatique, Faculté des sciences, Université de Sherbrooke, Québec

Types



 © 2018-2021, Μήτις - CC BY-NC-SA 4.0
Département d'informatique, Faculté des sciences, Université de Sherbrooke, Québec

TYPE
Motivation

- Permettre la création de nouveaux types de base.
- Faciliter la définition et l'évolution de schémas (particulièrement ceux de taille moyenne ou grande).
- Encapsuler la définition de contraintes internes entre des attributs qui ne peuvent être interprétés indépendamment (et qui de ce fait forment une représentation d'une même valeur).
  - En particulier, faciliter le maintien de la première forme normale.

SQL_02 : LDD Types et domaines (v210a) © 2018-2021, Μήτις - CC BY-NC-SA 4.0
Département d'informatique, Faculté des sciences, Université de Sherbrooke, Québec

Faire la représentation graphique au tableau

TYPE
Syntaxe ISO – un cas simple : l'agrégat

create_composite_type ::=
  CREATE TYPE *name* **AS (** [*attribute* [, …]] **)**

attribute ::=
  attribute_name data_type [**COLLATE** *collation*]

2021-11-01 SQL_02 : LDD Types et domaines (v210a) © 2018-2021, Μῆτις - CC BY-NC-SA 4.0
Département d'informatique, Faculté des sciences, Université de Sherbrooke, Québec

CREATE TYPE ***name*** AS
  ([***attribute_name data_type*** [COLLATE ***collation***] [, ...]])

CREATE TYPE ***name*** AS ENUM
  (['***label***' [, ...]])

CREATE TYPE ***name*** AS RANGE (
  SUBTYPE = ***subtype***
  [, SUBTYPE_OPCLASS = ***subtype_operator_class***]
  [, COLLATION = ***collation***]
  [, CANONICAL = ***canonical_function***]
  [, SUBTYPE_DIFF = ***subtype_diff_function***]
)

CREATE TYPE ***name*** (
  INPUT = ***input_function***,
  OUTPUT = ***output_function***
  [, RECEIVE = ***receive_function***]

[, SEND = ***send_function***]
[, TYPMOD_IN = ***type_modifier_input_function***]
[, TYPMOD_OUT = ***type_modifier_output_function***]
[, ANALYZE = ***analyze_function***]
[, INTERNALLENGTH = { ***internallength*** | VARIABLE }]
[, PASSEDBYVALUE]
[, ALIGNMENT = ***alignment***]
[, STORAGE = ***storage***]
[, LIKE = ***like_type***]
[, CATEGORY = ***category***]
[, PREFERRED = ***preferred***]
[, DEFAULT = ***default***]
[, ELEMENT = ***element***]
[, DELIMITER = ***delimiter***]
[, COLLATABLE = ***collatable***]
)

CREATE TYPE ***name***

TYPE
Syntaxe ISO – autres cas

- Plusieurs autres constructeurs de types sont définis par le standard, mais
  - ils sont rarement offerts par les dialectes SQL contemporains
  - lorsqu'ils le sont, leur syntaxe et leur sémantique sont généralement différentes de celle décrite dans le standard ISO

2021-11-01 SQL_02 : LDD Types et domaines (v210a) © 2018-2021, Μήτις - CC BY-NC-SA 4.0
Département d'informatique, Faculté des sciences, Université de Sherbrooke, Québec

TYPE
Syntaxe ISO (1/3)

```
<user-defined type definition> ::=
 CREATE TYPE <user-defined type body>

<user-defined type body> ::=
 <schema-resolved user-defined type name>
   [ <subtype clause> ]
   [ AS <representation> ]
   [ <user-defined type option list> ]
   [ <method specification list> ]

<user-defined type option list> ::=
 <user-defined type option>
 [ <user-defined type option>... ]

<user-defined type option> ::=
   <instantiable clause>
 | <finality>
 | <reference type specification>
 | <cast to ref>
 | <cast to type>
 | <cast to distinct>
 | <cast to source>

<subtype clause> ::=
 UNDER <supertype name>

<supertype name> ::=
 <path-resolved user-defined type name>

<representation> ::=
   <predefined type>
 | <collection type>
 | <member list>

<member list> ::=
 ( <member> [ { , <member> }... ] )

<member> ::=
 <attribute definition>

<instantiable clause> ::=
   INSTANTIABLE
 | NOT INSTANTIABLE
```

2021-11-01 SQL_02 : LDD Types et domaines (v210a) © 2018-2021, Μῆτις - CC BY-NC-SA 4.0
Département d'informatique, Faculté des sciences, Université de Sherbrooke, Québec

TYPE
Syntaxe ISO (2/3)

<finality> ::=
FINAL | NOT FINAL

<reference type specification> ::=
<user-defined representation>
| <derived representation>
| <system-generated representation>

<user-defined representation> ::=
REF USING <predefined type>

<derived representation> ::=
REF FROM <list of attributes>

<system-generated representation> ::=
REF IS SYSTEM GENERATED

<cast to ref> ::=
CAST (SOURCE AS REF)
WITH <cast to ref identifier>

<cast to ref identifier> ::=
<identifier>

<cast to type> ::=
CAST (REF AS SOURCE)
WITH <cast to type identifier>

<cast to type identifier> ::=
<identifier>

<list of attributes> ::=
(<attribute name> [{ , <attribute name> }...])

<cast to distinct> ::=
CAST (SOURCE AS DISTINCT)
WITH <cast to distinct identifier>

<cast to distinct identifier> ::=
<identifier>

<cast to source> ::=
CAST (DISTINCT AS SOURCE)
WITH <cast to source identifier>

<cast to source identifier> ::=
<identifier>

© 2018-2021, Mηπς - CC BY-NC-SA 4.0

TYPE
Syntaxe ISO (3/3)

<method specification list> ::=
<method specification>
[{ <comma> <method specification> }...]
<method specification> ::=
<original method specification>
| <overriding method specification>
<original method specification> ::=
<partial method specification>
[SELF AS RESULT]
[SELF AS LOCATOR]
[<method characteristics>]
<overriding method specification> ::=
OVERRIDING
<partial method specification>

<partial method specification> ::=
[INSTANCE | STATIC | CONSTRUCTOR]
METHOD <method name>
<SQL parameter declaration list>
<returns clause>
[SPECIFIC <specific method name>]
<specific method name> ::=
[<schema name> <period>]
<qualified identifier>
<method characteristics> ::=
<method characteristic>...
<method characteristic> ::=
<language clause>
| <parameter style clause>
| <deterministic characteristic>
| <SQL-data access indication>
| <null-call clause>

2021-11-01 SQL_02 : LDD Types et domaines (v210a) © 2018-2021, Μῆτις - CC BY-NC-SA 4.0
Département d'informatique, Faculté des sciences, Université de Sherbrooke, Québec

TYPE
Spécificité PostgreSQL – l'agrégat

- Comme pour ISO... ou presque!
- Exemple

```
type point_3D as
  ( x float, y float, z float )

create domain octant_3D_pos as
  point_3D    -- refusé par PostgreSQL 9.5;
              -- seuls les types de base prédéfinis sont autorisés
  check ((value).x >= 0.0 and (value).y >= 0.0 and (value).z >= 0.0)

create table loc_3D (
  id int,
  description text,
  pos point_3D,
  check ((pos).x >= 0.0 and (pos).y >= 0.0 and (pos).z >= 0.0)
)
```

2021-11-01 SQL_02 : LDD Types et domaines (v210a) © 2018-2021, Mητις - CC BY-NC-SA 4.0
Département d'informatique, Faculté des sciences, Université de Sherbrooke, Québec

CREATE TYPE ***name*** AS
  ([***attribute_name data_type*** [COLLATE ***collation***] [, ...]])

CREATE TYPE ***name*** AS ENUM
  (['***label***' [, ...]])

CREATE TYPE ***name*** AS RANGE (
  SUBTYPE = ***subtype***
  [, SUBTYPE_OPCLASS = ***subtype_operator_class***]
  [, COLLATION = ***collation***]
  [, CANONICAL = ***canonical_function***]
  [, SUBTYPE_DIFF = ***subtype_diff_function***]
)

CREATE TYPE ***name*** (
  INPUT = ***input_function***,
  OUTPUT = ***output_function***
  [, RECEIVE = ***receive_function***]

[, SEND = ***send_function***]
[, TYPMOD_IN = ***type_modifier_input_function***]
[, TYPMOD_OUT = ***type_modifier_output_function***]
[, ANALYZE = ***analyze_function***]
[, INTERNALLENGTH = { ***internallength*** | VARIABLE }]
[, PASSEDBYVALUE]
[, ALIGNMENT = ***alignment***]
[, STORAGE = ***storage***]
[, LIKE = ***like_type***]
[, CATEGORY = ***category***]
[, PREFERRED = ***preferred***]
[, DEFAULT = ***default***]
[, ELEMENT = ***element***]
[, DELIMITER = ***delimiter***]
[, COLLATABLE = ***collatable***]
)

CREATE TYPE ***name***

TYPE

Spécificité PostgreSQL – l'énumération

CREATE TYPE ***name*** AS ENUM (['***label***' [, ...]])

SQL_02 : LDD Types et domaines (v210a) © 2018-2021, Μῆτις - CC BY-NC-SA 4.0
Département d'informatique, Faculté des sciences, Université de Sherbrooke, Québec

CREATE TYPE ***name*** AS
([***attribute_name data_type*** [COLLATE ***collation***] [, ...]])

CREATE TYPE ***name*** AS ENUM
(['***label***' [, ...]])

CREATE TYPE ***name*** AS RANGE (
SUBTYPE = ***subtype***
[, SUBTYPE_OPCLASS = ***subtype_operator_class***]
[, COLLATION = ***collation***]
[, CANONICAL = ***canonical_function***]
[, SUBTYPE_DIFF = ***subtype_diff_function***]
)

CREATE TYPE ***name*** (
INPUT = ***input_function***,
OUTPUT = ***output_function***
[, RECEIVE = ***receive_function***]

    [, SEND = ***send_function***]
    [, TYPMOD_IN = ***type_modifier_input_function***]
    [, TYPMOD_OUT = ***type_modifier_output_function***]
    [, ANALYZE = ***analyze_function***]
    [, INTERNALLENGTH = { ***internallength*** | VARIABLE }]
    [, PASSEDBYVALUE]
    [, ALIGNMENT = ***alignment***]
    [, STORAGE = ***storage***]
    [, LIKE = ***like_type***]
    [, CATEGORY = ***category***]
    [, PREFERRED = ***preferred***]
    [, DEFAULT = ***default***]
    [, ELEMENT = ***element***]
    [, DELIMITER = ***delimiter***]
    [, COLLATABLE = ***collatable***]
)

CREATE TYPE ***name***

TYPE
Spécificité PostgreSQL – l'intervalle

CREATE TYPE ***name*** AS RANGE
(
SUBTYPE = ***subtype***
[, SUBTYPE_OPCLASS = ***subtype_operator_class***]
[, COLLATION = ***collation***]
[, CANONICAL = ***canonical_function***]
[, SUBTYPE_DIFF = ***subtype_diff_function***]
)

2021-11-01 SQL_02 : LDD Types et domaines (v210a) © 2018-2021, Μῆτις - CC BY-NC-SA 4.0
Département d'informatique, Faculté des sciences, Université de Sherbrooke, Québec

CREATE TYPE ***name*** AS
([***attribute_name data_type*** [COLLATE ***collation***] [, ...]])

CREATE TYPE ***name*** AS ENUM
(['***label***' [, ...]])

CREATE TYPE ***name*** AS RANGE (
SUBTYPE = ***subtype***
[, SUBTYPE_OPCLASS = ***subtype_operator_class***]
[, COLLATION = ***collation***]
[, CANONICAL = ***canonical_function***]
[, SUBTYPE_DIFF = ***subtype_diff_function***]
)

CREATE TYPE ***name*** (
INPUT = ***input_function***,
OUTPUT = ***output_function***
[, RECEIVE = ***receive_function***]

[, SEND = ***send_function***]
[, TYPMOD_IN = ***type_modifier_input_function***]
[, TYPMOD_OUT = ***type_modifier_output_function***]
[, ANALYZE = ***analyze_function***]
[, INTERNALLENGTH = { ***internallength*** | VARIABLE }]
[, PASSEDBYVALUE]
[, ALIGNMENT = ***alignment***]
[, STORAGE = ***storage***]
[, LIKE = ***like_type***]
[, CATEGORY = ***category***]
[, PREFERRED = ***preferred***]
[, DEFAULT = ***default***]
[, ELEMENT = ***element***]
[, DELIMITER = ***delimiter***]
[, COLLATABLE = ***collatable***]
)

CREATE TYPE ***name***

TYPE

Spécificité PostgreSQL – la définition externe

```
CREATE TYPE name
  (
  INPUT = input_function,
  OUTPUT = output_function
  [ , RECEIVE = receive_function ]
  [ , SEND = send_function ]
  [ , TYPMOD_IN = type_modifier_input_function ]
  [ , TYPMOD_OUT = type_modifier_output_function ]
  [ , ANALYZE = analyze_function ]
  [ , INTERNALLENGTH = { internallength | VARIABLE } ]
  [ , PASSEDBYVALUE ]
  [ , ALIGNMENT = alignment ]
  [ , STORAGE = storage ]
  [ , LIKE = like_type ]
  [ , CATEGORY = category ]
  [ , PREFERRED = preferred ]
  [ , DEFAULT = default ]
  [ , ELEMENT = element ]
  [ , DELIMITER = delimiter ]
  [ , COLLATABLE = collatable ]
  )
```

2021-11-01 SQL_02 : LDD Types et domaines (v210a) © 2018-2021, Mητις - CC BY-NC-SA 4.0
Département d'informatique, Faculté des sciences, Université de Sherbrooke, Québec

Syntaxe ISO

<user-defined type definition> ::= CREATE TYPE <user-defined type body> <user-defined type body> ::= <schema-resolved user-defined type name> [<subtype clause>] [AS <representation>] [<user-defined type option list>] [<method specification list>] <user-defined type option list> ::= <user-defined type option> [<user-defined type option>...] <user-defined type option> ::= <instantiable clause> | <finality> | <reference type specification> | <cast to ref> | <cast to type> | <cast to distinct> | <cast to source> <subtype clause> ::= UNDER <supertype name> <supertype name> ::= <path-resolved user-defined type name> <representation> ::= <predefined type> | <collection type> | <member list> <member list> ::= <left paren> <member> [{ <comma> <member> }...] <right paren> <member> ::= <attribute definition> <instantiable clause> ::= INSTANTIABLE | NOT INSTANTIABLE

13

716 Foundation (SQL/Foundation)

©ISO/IEC 2010 – All rights reserved

<finality> ::= FINAL | NOT FINAL

<reference type specification> ::= <user-defined representation> | <derived representation> | <system-generated representation> <user-defined representation> ::= REF USING <predefined type> <derived representation> ::= REF FROM <list of

attributes> <system-generated representation> ::= REF IS SYSTEM GENERATED

<cast to ref> ::= CAST <left paren> SOURCE AS REF <right paren> WITH <cast to ref identifier> <cast to ref identifier> ::= <identifier> <cast to type> ::= CAST <left paren> REF AS SOURCE <right paren> WITH <cast to type identifier> <cast to type identifier> ::= <identifier> <list of attributes> ::= <left paren> <attribute name> [{ <comma> <attribute name> }...] <right paren> <cast to distinct> ::= CAST <left paren> SOURCE AS DISTINCT <right paren> WITH <cast to distinct identifier> <cast to distinct identifier> ::= <identifier> <cast to source> ::= CAST <left paren> DISTINCT AS SOURCE <right paren> WITH <cast to source identifier> <cast to source identifier> ::= <identifier> <method specification list> ::= <method specification> [{ <comma> <method specification> }...] <method specification> ::= <original method specification> | <overriding method specification> <original method specification> ::= <partial method specification> [SELF AS RESULT] [SELF AS LOCATOR] [<method characteristics>] <overriding method specification> ::= OVERRIDING <partial method specification>

©ISO/IEC 2010 – All rights reserved

<partial method specification> ::= [INSTANCE | STATIC | CONSTRUCTOR] METHOD <method name> <SQL parameter declaration list> <returns clause> [SPECIFIC <specific method name>] <specific method name> ::= [<schema name> <period>]<qualified identifier> <method characteristics> ::= <method characteristic>... <method characteristic> ::= <language clause> | <parameter style clause> | <deterministic characteristic> | <SQL-data access indication> | <null-call clause>

TYPE
Transportabilité

- Syntaxe et sémantique ISO généralement non respectées d'un dialecte à l'autre.
- Syntaxe et sémantique variables d'un dialecte à l'autre.

SQL_02 : LDD Types et domaines (v210a) © 2018-2021, Μήτις - CC BY-NC-SA 4.0
Département d'informatique, Faculté des sciences, Université de Sherbrooke, Québec

Faire la représentation graphique au tableau

Domaines et types
L'éditorial

- Le bon usage des domaines (types) est fondamental.
- La mauvaise volonté évidente des éditeurs à mettre en oeuvre une solution standard, donc transportable, rend cette très bonne pratique plutôt difficile… en pratique!

2021-11-01 SQL_02 : LDD Types et domaines (v210a) © 2018-2021, Μητις - CC BY-NC-SA 4.0
Département d'informatique, Faculté des sciences, Université de Sherbrooke, Québec

SQL ISO
L’éditorial

- *ISO or not ISO, that’s the question !*
- *Is it ?*

- On invoque souvent la taille et l’incohérence de la norme ainsi que les problèmes de performance que pourrait entrainer l’adhésion à certaines exigences (comme si un résultat rapide, mais faux était préférable).
- Il y a certes matière à réduire et épurer le langage, voire à en définir un nouveau. En attendant que cela soit fait, il serait avantageux pour tous (développeurs, informaticiens et maitres d’ouvrage) que les éditeurs adhèrent strictement à la norme.

2021-11-01 SQL_02 : LDD Types et domaines (v210a) © 2018-2021, Μήτις - CC BY-NC-SA 4.0
Département d'informatique, Faculté des sciences, Université de Sherbrooke, Québec

Faire la représentation graphique au tableau

Références

- Loney, Kevin ;
  Oracle Database 11g: The Complete Reference.
  Oracle Press/McGraw-Hill/Osborne, 2008.
  ISBN 978-0071598750.
- Date, Chris J. ;
  SQL and Relational Theory: How to Write Accurate SQL Code.
  2nd edition, O'Reilly, 2012.
  ISBN 978-1-449-31640-2.
- Le site d'Oracle (en anglais)
  - http://docs.oracle.com/cd/E11882_01/index.htm
- Le site de PostgreSQL (en français)
  - http://docs.postgresqlfr.org

2021-11-01 SQL_02 : LDD Types et domaines (v210a) © 2018-2021, Μῆτις - CC BY-NC-SA 4.0
Département d'informatique, Faculté des sciences, Université de Sherbrooke, Québec