Bases de données
SQL

LDD
Types et domaines

SQL_02
v210a

2021-11-01

UNIVERSITÉ DE SHERBROOKE

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

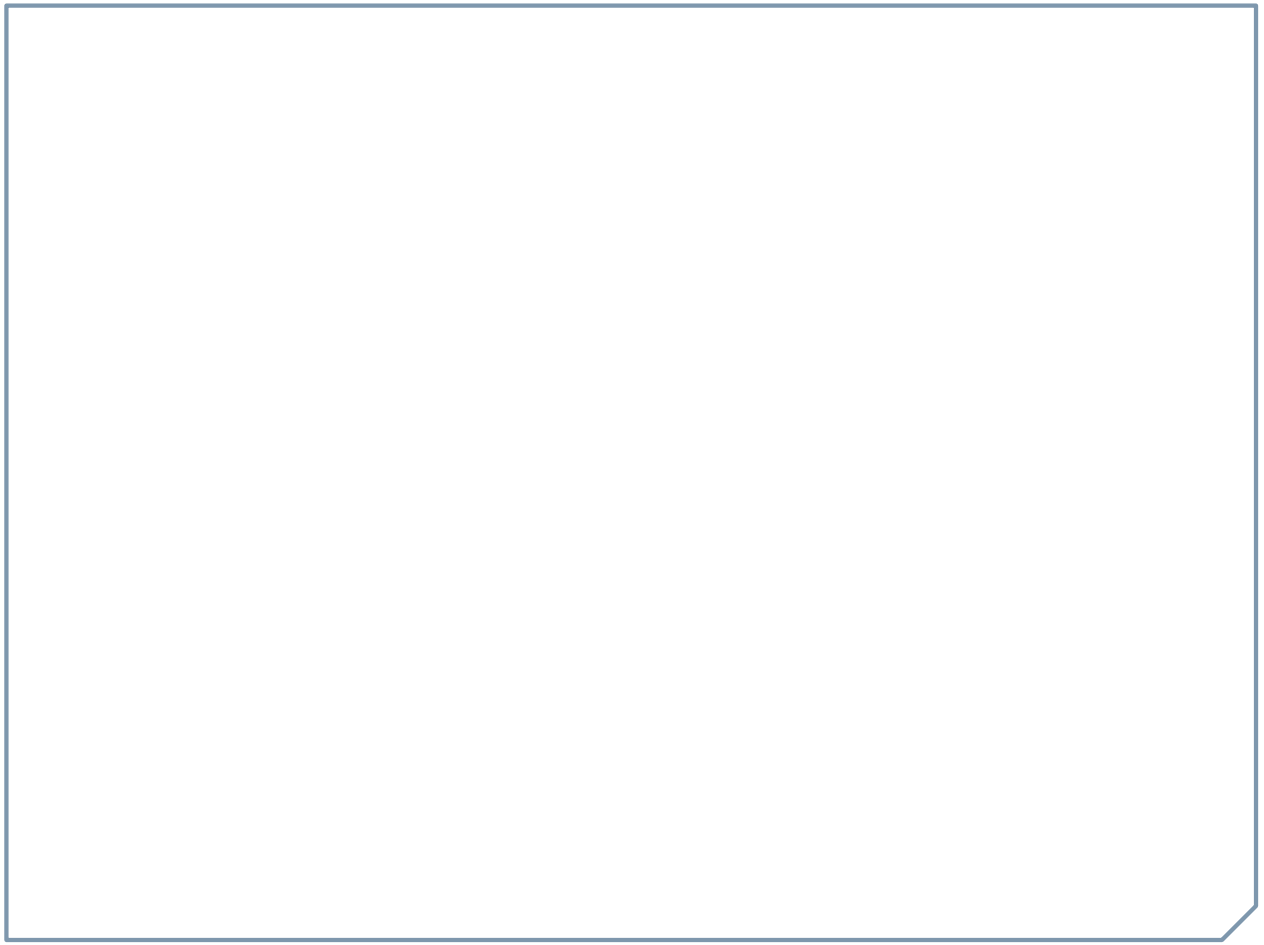
Plan

- 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

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.***

Domaines



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.

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>]

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 }

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);
```

DOMAIN
Transportabilité

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

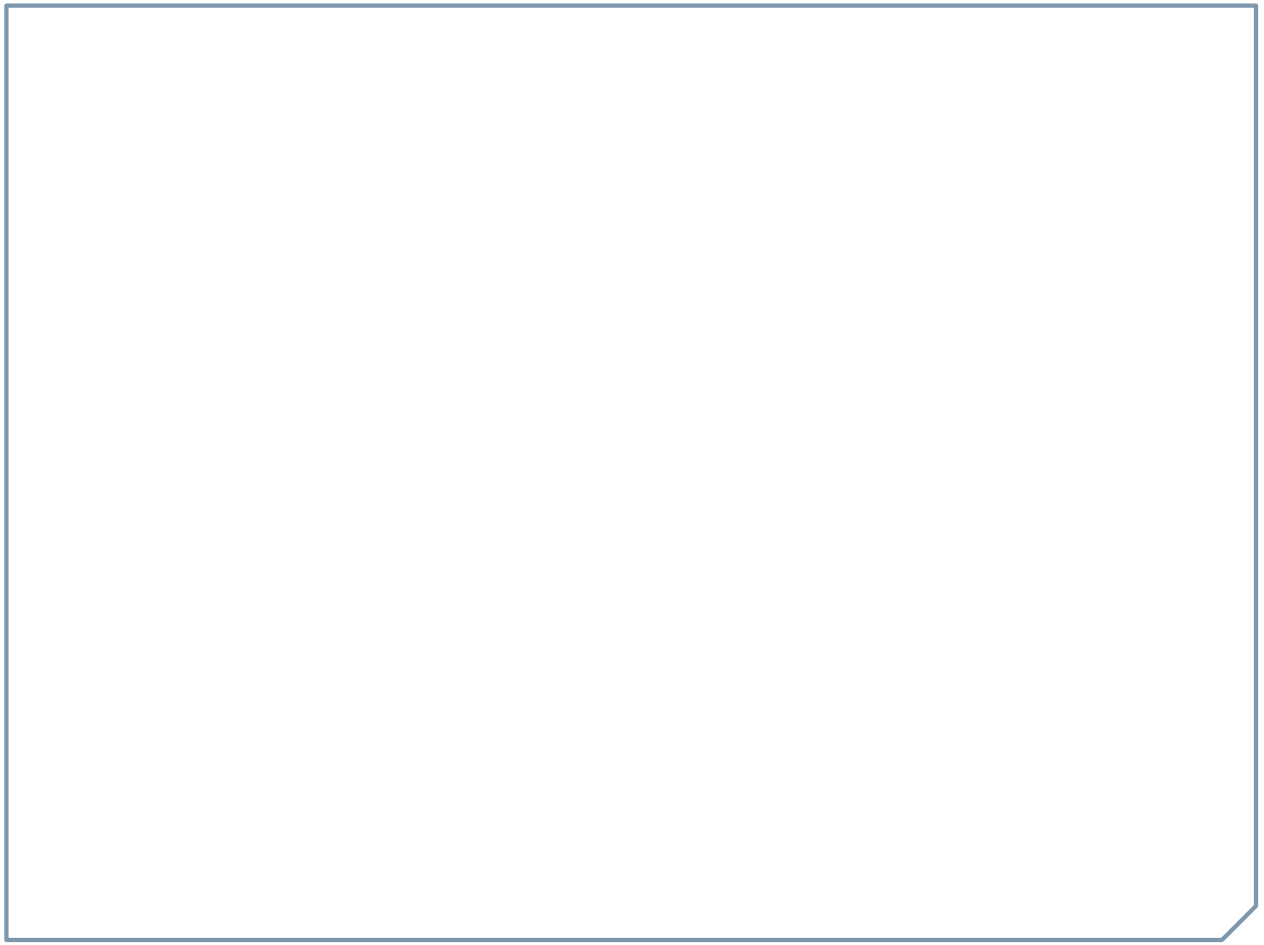
  CREATE TYPE

  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.

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.

Types



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.

TYPE
Syntaxe ISO – un cas simple : l'agrégat

create_composite_type ::=

CREATE TYPE *name* **AS (** [*attribute* [, ...]] **)**

attribute ::=

attribute_name *data_type* [**COLLATE** *collation*]

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

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

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>

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>

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)
)
```

TYPE

Spécificité PostgreSQL – l'énumération

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

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***]

)

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***]
)

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.

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!

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.

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