Bases de données
SQL

Types élémentaires et prédéfinis

SQL_01
v401b

2022-01-25

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

- Types prédéfinis usuels
- Types prédéfinis de stockage [*]
- Définition de types
  - sous-types : DOMAIN
  - types de base : TYPE [*]
- Exercices
- Références

[*] sujet pouvant être différé dans un premier temps

2022-01-25 SQL_01 - Types élémentaires (v401b) © 2018-2022, Μῆτις - CC BY-NC-SA 4.0
Département d'informatique, Faculté des sciences, Université de Sherbrooke, Québec

2

Types prédéfinis usuels

La norme ISO prévoit une riche palette de types prédéfinis.

Si PostgreSQL adhère assez bien à la norme, plusieurs dialectes s'en écartent, parfois même significativement.

- Types prédéfinis en SQL
  - Type booléen
  - Types textuels
  - Types numériques
  - Types temporels
- Types prédéfinis en PostgreSQL
  - Type booléen
  - Types textuels
  - Types numériques
  - Types temporels

2022-01-25 SQL_01 - Types élémentaires (v401b) © 2018-2022, Μῆτις - CC BY-NC-SA 4.0
Département d'informatique, Faculté des sciences, Université de Sherbrooke, Québec

Types prédéfinis ISO

2022-01-25 SQL_01 - Types élémentaires (v401b) © 2018-2022, Μῆτις - CC BY-NC-SA 4.0
Département d'informatique, Faculté des sciences, Université de Sherbrooke, Québec

4

Types prédéfinis ISO (présents depuis ISO 9075:2003)

Les nombres	Les autres
SMALLINT	BOOLEAN
INTEGER	
BIGINT	CHARACTER (n) CHAR (n)
NUMERIC (p,s)	CHARACTER VARYING (n) VARCHAR (n)
DECIMAL (p,s)	
FLOAT (p)	DATE
REAL	TIME (p)
DOUBLE PRECISION	TIMESTAMP (p)
	INTERVAL (p)

2022-01-25 SQL_01 - Types élémentaires (v401b) © 2018-2022, Μῆτις - CC BY-NC-SA 4.0
Département d'informatique, Faculté des sciences, Université de Sherbrooke, Québec

5

La différence entre les trois types entiers est donnée par smallint ⊆ integer ⊆ bigint. Quant à leur cardinalité minimale, elle a varié dans le temps. La plupart des SGBDR les traitent identiquement dont Oracle, PostgreSQL et MS-SQL.

La différence entre numeric et decimal : numeric doit fournir EXACTEMENT la précision demandée, decimal AU MOINS la précision demandée.

ISO/IEC 9075-2:2003 (E) 6.1 <data type> pp 164-165

18) Octets in a binary large object string are numbered beginning with 1 (one). The length in octets of the string is variable, with a minimum length of 0 (zero) and a maximum length of the value of <large object length>.

19) The <scale> of an <exact numeric type> shall not be greater than the <precision> of the <exact numeric type>.

20) For the <exact numeric type>s DECIMAL and NUMERIC:

a) The maximum value of <precision> is implementation-defined. <precision> shall not be greater than this value.

b) The maximum value of <scale> is implementation-defined. <scale> shall not be greater than this maximum value.

21) NUMERIC specifies the data type exact numeric, with the decimal precision and scale specified by the <precision> and <scale>.

22) DECIMAL specifies the data type exact numeric, with the decimal scale specified by the <scale> and the implementation-defined decimal precision equal to or greater than the value of the specified <precision>.

23) SMALLINT, INTEGER, and BIGINT specify the data type exact numeric, with scale of 0 (zero) and binary or decimal precision. The choice of binary versus decimal precision is implementation-defined, but the same radix shall be chosen for all three data types. The precision of SMALLINT shall be less than or equal to the precision of INTEGER, and the precision of BIGINT shall be greater than or equal to the precision of INTEGER.

24) FLOAT specifies the data type approximate numeric, with binary precision equal to or greater than the value of the specified <precision>. The maximum value of <precision> is implementation-defined. <precision> shall not be greater than this value.

25) REAL specifies the data type approximate numeric, with implementation-defined precision.

26) DOUBLE PRECISION specifies the data type approximate numeric, with implementation-defined precision that is

greater than the implementation-defined precision of REAL.

Types prédéfinis ISO : type booléen

- BOOLEAN (FALSE, TRUE, UNKNOWN)

2022-01-25 SQL_01 - Types élémentaires (v401b) © 2018-2022, Μῆτις - CC BY-NC-SA 4.0
Département d'informatique, Faculté des sciences, Université de Sherbrooke, Québec

Types prédéfinis ISO : types textuels

- CHARACTER : texte de longueur fixée et constante
- CHARACTER VARYING : texte de longueur variable

- CHARACTER s'abrège en CHAR
- CHARACTER VARYING s'abrège en VARCHAR

- Dans les deux cas, il **faut** de suffixer une limite entre parenthèses :
  - exacte dans le cas de CHAR
  - maximale dans le cas de VARCHAR

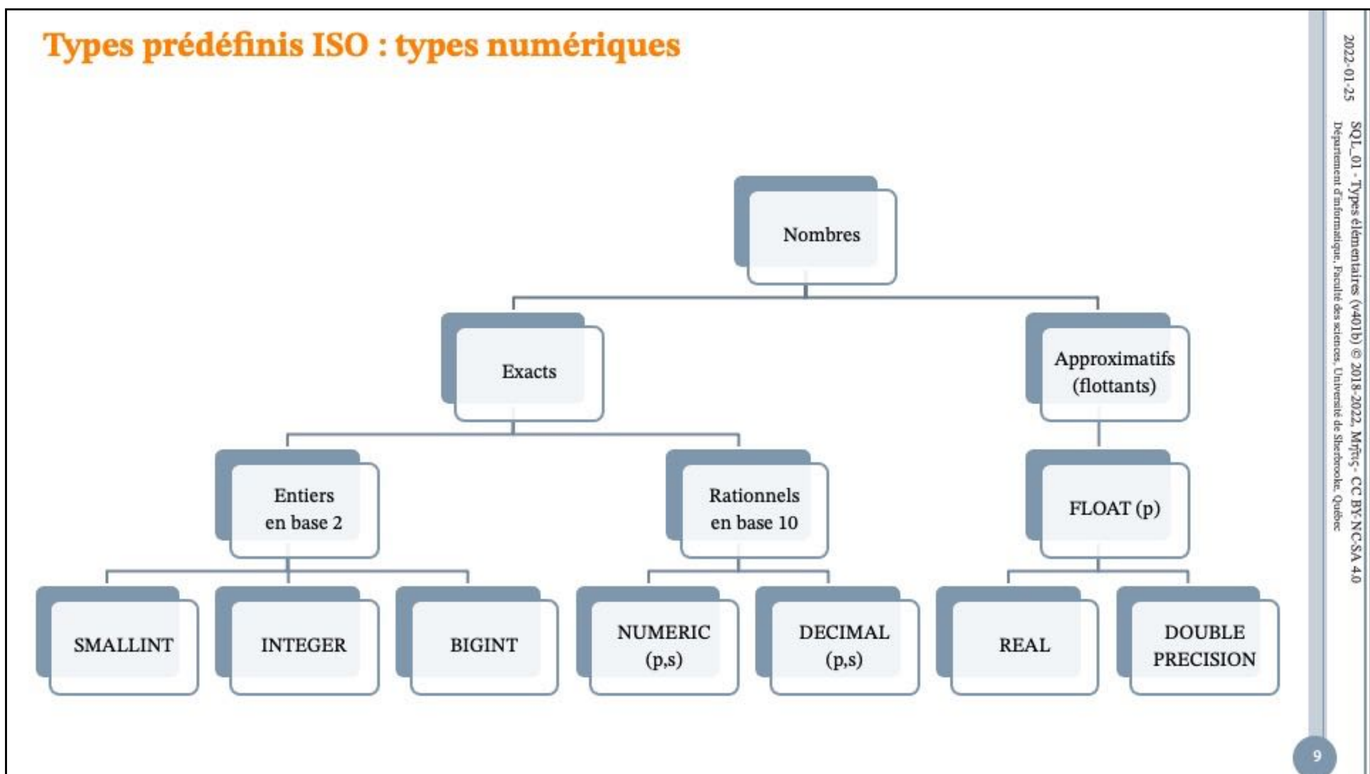
- Exemples
  - CHAR (12) – exactement 12 caractères
  - VARCHAR (12) – au plus 12 caractères

2022-01-25 SQL_01 - Types élémentaires (v401b) © 2018-2022, Μῆτις - CC BY-NC-SA 4.0
Département d'informatique, Faculté des sciences, Université de Sherbrooke, Québec

Types prédéfinis ISO : types textuels (recommandations)

- De l'incompatibilité des CHAR et VARCHAR
  - ...
- Une recommandation de plus en plus fréquente :
  - NE PAS UTILISER CHARACTER
  - Une exception généralement acceptée : les codes et matricules

2022-01-25 SQL_01 - Types élémentaires (v401b) © 2018-2022, Μῆτις - CC BY-NC-SA 4.0
Département d'informatique, Faculté des sciences, Université de Sherbrooke, Québec



C’est important de choisir le bon type et de restreindre le domaine de valeur au plus juste.

Types prédéfinis ISO types temporels

Le nécessaire

- p : un point sur l'axe du temps : TIMESTAMP
- i : un intervalle de points consécutifs : RANGE
- une durée : INTERVAL *(sic)*

L'utile

- Une référence à un point d'un calendrier : DATE
- Une référence à un moment de la journée : TIME

Perception continue

$-\infty$ $+\infty$

Modèle discret

α γ p durée i ω

d=date(p)

Modèle calendaire

ϕ d

α = point initial (minimum)
ω = point final (maximum)
γ = distance entre deux points consécutifs
ϕ = valeur calendaire associée à α

$$\text{durée}(i) = \text{card}(i) * \gamma$$

$$\text{date}(\alpha) = \phi$$

$$p_i < p_j \Rightarrow \text{date}(p_i) \leq \text{date}(p_j)$$

$$d_k < d_\ell \Rightarrow \text{date}^{-1}(d_k) \leq \text{date}^{-1}(d_\ell)$$

2022-01-25 SQL_01 - Types élémentaires (v401b) © 2018-2022, Μῆτις - CC BY-NC-SA 4.0
Département d'informatique, Faculté des sciences, Université de Sherbrooke, Québec

10

time, timestamp, and interval accept an optional precision value p which specifies the number of fractional digits retained in the seconds field. By default, there is no explicit bound on precision. The allowed range of p is from 0 to 6 for the timestamp and interval types.

WITH TIME ZONE : relatif au UTC

Exemples :

date '2015-09-01'
time '10:30'

timestamp '2015-09-01 10:30:00'

https://docs.postgresql.fr/14/datatype-datetime.html

https://docs.postgresql.fr/14/functions-datetime.html

Types prédéfinis PostgreSQL

2022-01-25 SQL_01 - Types élémentaires (v401b) © 2018-2022, Μῆτις - CC BY-NC-SA 4.0
Département d'informatique, Faculté des sciences, Université de Sherbrooke, Québec

Types prédéfinis PostgreSQL : type booléen

Nom	Description
`boolean`	état vrai ou faux

« Ce type dispose de plusieurs états : *true* (vrai), *false* (faux) et un troisième état, *unknown* (inconnu), qui est représenté par la valeur SQL NULL [sic] ».

https://docs.postgresql.fr/current/datatype-boolean.html

2022-01-25 SQL_01 - Types élémentaires (v401b) © 2018-2022, Μήτις - CC BY-NC-SA 4.0
Département d'informatique, Faculté des sciences, Université de Sherbrooke, Québec

12

Valid literal values for the "true" state are:

TRUE

't'

'true'

'y'

'yes'

'on'

'1'

For the "false" state, the following values can be used:

FALSE

'f'

'false'

'n'

'no'

'off'

'0'

Leading or trailing whitespace is ignored, and case does not matter. The key words TRUE and FALSE are the preferred (SQL-compliant) usage.

Types prédéfinis PostgreSQL : types textuels

Nom	Synonyme	Description
character varying(n)	varchar(n)	longueur variable avec limite
character(n)	char(n)	longueur fixe, complété par des espaces
text		longueur variable illimitée

https://docs.postgresql.fr/current/datatype-boolean.html

2022-01-25 SQL_01 - Types élémentaires (v401b) © 2018-2022, Μῆτις - CC BY-NC-SA 4.0
Département d'informatique, Faculté des sciences, Université de Sherbrooke, Québec

13

SQL définit deux types de caractères principaux : character varying(***n***) et character(***n***) où ***n*** est un entier positif. Ces deux types permettent de stocker des chaînes de caractères de taille inférieure ou égale à ***n*** (ce ne sont pas des octets). Toute tentative d'insertion d'une chaîne plus longue conduit à une erreur, à moins que les caractères en excès ne soient tous des espaces, auquel cas la chaîne est tronquée à la taille maximale (cette exception étrange est imposée par la norme SQL). Si la chaîne à stocker est plus petite que la taille déclarée, les valeurs de type character sont complétées par des espaces, celles de type character varying sont stockées en l'état.

Si une valeur est explicitement transtypée en character varying(***n***) ou en character(***n***), une valeur trop longue est tronquée à ***n*** caractères sans qu'aucune erreur ne soit levée (ce comportement est aussi imposé par la norme SQL.)

Les notations varchar(***n***) et char(***n***) sont des alias de character varying(***n***) et character(***n***), respectivement. character sans indication de taille est équivalent à character(1). Si character varying est utilisé sans indicateur de taille, le type accepte des chaînes de toute taille. Il s'agit là d'une spécificité dePostgreSQL.

De plus, PostgreSQL propose aussi le type text, qui permet de stocker des chaînes de n'importe quelle taille. Bien que le type text ne soit pas dans le standard SQL, plusieurs autres systèmes de gestion de bases de données SQL le proposent également.

Les valeurs de type character sont complétées physiquement à l'aide d'espaces pour atteindre la longueur ***n*** indiquée. Ces valeurs sont également stockées et affichées de cette façon. Cependant, les espaces de remplissage sont traités comme sémantiquement non significatifs et sont donc ignorés lors de la comparaison de deux valeurs de type character. Dans les collationnements où les espaces de remplissage sont significatifs, ce comportement peut produire des résultats inattendus, par exemple SELECT 'a '::CHAR(2) collate "C" < E'a\n'::CHAR(2) retourne vrai, même si la locale C considérerait qu'un espace est plus grand qu'un retour chariot. Les espaces de

remplissage sont supprimés lors de la conversion d'une valeur character vers l'un des autres types chaîne. Ces espaces *ont* une signification sémantique pour les valeurs de type character varying et text, et lors de l'utilisation de la correspondance de motifs, par exemple avec LIKE ou avec les expressions rationnelles.

Les caractères pouvant être enregistrés dans chacun de ces types de données sont déterminés par le jeu de caractères de la base de données, qui a été sélectionné à la création de la base. Quelque soit le jeu de caractères spécifique, le caractère de code zéro (quelque fois appelé NUL) ne peut être enregistré. Pour plus d'informations, voir **Section 24.3**.

L'espace nécessaire pour une chaîne de caractères courte (jusqu'à 126 octets) est de un octet, plus la taille de la chaîne qui inclut le remplissage avec des espaces dans le cas du type character. Les chaînes plus longues ont quatre octets d'en-tête au lieu d'un seul. Les chaînes longues sont automatiquement compressées par le système, donc le besoin pourrait être moindre. Les chaînes vraiment très longues sont stockées dans des tables supplémentaires, pour qu'elles n'empêchent pas d'accéder rapidement à des valeurs plus courtes. Dans tous les cas, la taille maximale possible pour une chaîne de caractères est de l'ordre de 1 Go. (La taille maximale pour ***n*** dans la déclaration de type est inférieure. Il ne sert à rien de modifier ce comportement, car avec les encodages sur plusieurs octets, les nombres de caractères et d'octets peuvent être très différents. Pour stocker de longues chaînes sans limite supérieure précise, il est préférable d'utiliser les types text et character varying sans taille, plutôt que d'indiquer une limite de taille arbitraire.)

Types prédéfinis PostgreSQL : types numériques

Nom	T	Description	Étendue
`smallint`	2	entier de faible étendue	de -32768 à +32767
`integer`	4	entier habituelr	de -2147483648 à +2147483647
`bigint`	8	grand entier	de -9223372036854775808 à +9223372036854775807
`decimal`	v	précision indiquée par l'utilisateur, valeur exacte	jusqu'à 131072 chiffres avant « . » jusqu'à 16383 chiffres après « . »
`numeric`	v	précision indiquée par l'utilisateur, valeur exacte	jusqu'à 131072 chiffres avant « . » jusqu'à 16383 chiffres après « . »
`real`	4	précision variable, valeur inexacte	précision de 6 décimales
`double precision`	8	précision variable, valeur inexacte	précision de 15 décimales
`smallserial`	2	entier à incrémentation automatique	de 1 à 32767
`serial`	4	entier à incrémentation automatique	de 1 à 2147483647
`bigserial`	8	entier à incrémentation automatique	de 1 à 9223372036854775807

https://www.postgresql.org/docs/current/static/datatype-numeric.html

2022-01-25 SQL_01 - Types élémentaires (v401b) © 2018-2022, Μῆτις - CC BY-NC-SA 4.0
Département d'informatique, Faculté des sciences, Université de Sherbrooke, Québec
14

T représente la taille en octets; la mention v signifie que c'est variable en fonction de la « longueur » de la valeur.

The maximum allowed precision when explicitly specified in the type declaration is 1000; NUMERIC without a specified precision is subject to the limits described in Table 8-2.

On most platforms, the real type has a range of at least 1E-37 to 1E+37 with a precision of at least 6 decimal digits. The double precision type typically has a range of around 1E-307 to 1E+308 with a precision of at least 15 digits. Values that are too large or too small will cause an error. Rounding might take place if the precision of an input number is too high. Numbers too close to zero that are not representable as distinct from zero will cause an underflow error.

Types prédéfinis PostgreSQL : types temporels — point, date et heure

2022-01-25

Nom	T	Description	Min	Max	Résolution
timestamp [(*p*)] [without time zone]	8	date et heure sans fuseau horaire	4713 BC	294 276 AD	1 microseconde / 14 chiffres
timestamp [(*p*)] with time zone	8	date et heure avec fuseau horaire	4713 BC	294 276 AD	1 microseconde / 14 chiffres
date	4	date seule (pas d'heure)	4713 BC	5 874 897 AD	1 jour
time [(*p*)] [without time zone]	8	heure seule sans fuseau horaire	00:00:00	24:00:00	1 microseconde / 14 chiffres
time [(*p*)] with time zone	12	heure seule avec fuseau horaire	00:00:00+1559	24:00:00-1559	1 microseconde / 14 chiffres

La précision (p) prescrit que la plus petite fraction de seconde représentable soit 10^{-p}.

https://docs.postgresql.fr/current/datatype-datetime.html

Département d'informatique, Faculté des sciences, Université de Sherbrooke, Québec

Ldes types time, timestamp, et interval acceptent une précision optionnelle ***p***, qui indique le nombre de décimales pour les secondes. Il n'y a pas, par défaut, de limite explicite à cette précision. Les valeurs acceptées pour ***p*** s'étendent de 0 à 6.

WITH TIME ZONE : relatif au UTC

Exemples :

date '2015-09-01'
time '10:30'
timestamp '2015-09-01 10:30:00'

AVERTISSEMENT : le teste suivant est faux :

«Pour timestamp with time zone, la valeur stockée en interne est toujours en UTC (*Universal Coordinated Time* ou Temps Universel Coordonné), aussi connu sous le nom de GMT (*Greenwich Mean Time*). Les valeurs saisies avec un fuseau horaire explicite sont converties en UTC à l'aide du décalage approprié. Si aucun fuseau horaire n'est précisé, alors le système considère que la date est dans le fuseau horaire indiqué par le paramètre système **TimeZone**, et la convertit en UTC en utilisant le décalage de la zone timezone. »

GMT et UTC sont différents! L'intégration de la dérive relative au temps solaire est différente : quotidienne pour GMT et annuelle pour UTC. Le modèle GMT a été remplacé par le modèle UTC comme référentiel officiel.

Types prédéfinis PostgreSQL : types temporels — durée

Nom	T	Description	Min	Max	Résolution
interval [*champs*] [(*p*)]	16	*durée*	-178 000 000 years	178 000 000 years	1 microsecond / 14 digits

La granularité (*champs*) est l'une des suivantes :

- YEAR
- MONTH
- DAY
- HOUR
- MINUTE
- SECOND
- YEAR TO MONTH
- DAY TO HOUR
- DAY TO MINUTE
- DAY TO SECOND
- HOUR TO MINUTE
- HOUR TO SECOND
- MINUTE TO SECOND

La précision (p) n'est significative que si la granularité comprend la seconde, elle prescrit alors que la plus petite fraction de seconde représentable soit 10^{-p}.

2022-01-25 SQL_01 - Types élémentaires (v401b) © 2018-2022, Μῆτις - CC BY-NC-SA 4.0
Département d'informatique, Faculté des sciences, Université de Sherbrooke, Québec

Notez que si ***champs*** et ***p*** sont tous les deux indiqués, ***champs*** doit inclure SECOND, puisque la précision s'applique uniquement aux secondes.

Le type time with time zone est défini dans le standard SQL, mais sa définition lui prête des propriétés qui font douter de son utilité. Dans la plupart des cas, une combinaison de date, time, timestamp without time zone et timestamp with time zone devrait permettre de résoudre toutes les fonctionnalités de date et heure nécessaires à une application.

Types prédéfinis de stockage

Les types de stockage sont en général à proscrire puisqu'ils ont pour effet de soustraire leurs valeurs à tout contrôle.

Leur usage prépondérant est le transport de données chiffrées.

Sujet pouvant être différé dans un premier temps

- Petits «objets»
  - Séquence de caractères.
  - Séquence d'octets.
- Grands «objets»
  - Séquence de caractères.
  - Séquence d'octets.

2022-01-25 SQL_01 - Types élémentaires (v401b) © 2018-2022, Μῆτις - CC BY-NC-SA 4.0
Département d'informatique, Faculté des sciences, Université de Sherbrooke, Québec

17

Types de stockage : petits objets

- Depuis 2006, les «séquences d'octets»
  - BINARY *(n)*
  - BINARY VARYING *(n)*
- Avant 2003, les «séquences de bits»
  - BIT *(n)*
  - BIT VARYING *(n)*

Types non utilisés en cours

2022-01-25 SQL_01 - Types élémentaires (v401b) © 2018-2022, Μῆτις - CC BY-NC-SA 4.0
Département d'informatique, Faculté des sciences, Université de Sherbrooke, Québec

18

Types de stockage : grands objets

- CHARACTER LARGE OBJECT *(n)* (CLOB)
- BINARY LARGE OBJECT *(n)* (BLOB)

Types non utilisés en cours

2022-01-25 SQL_01 - Types élémentaires (v401b) © 2018-2022, Μῆτις - CC BY-NC-SA 4.0
Département d'informatique, Faculté des sciences, Université de Sherbrooke, Québec

Déclaration de types

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'un 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.***

- DOMAIN (un survol)
  - Motivation
  - Syntaxe ISO
  - Spécificités PostgreSQL
  - Exemples
  - Transportabilité
  - Recommandations
- TYPE (un survol)
  - Motivation
  - Syntaxe ISO
  - Spécificités PostgreSQL
  - Exemples
  - Transportabilité
  - Recommandations

2022-01-25 SQL_01 - Types élémentaires (v401b) © 2018-2022, Μῆτις - CC BY-NC-SA 4.0
Département d'informatique, Faculté des sciences, Université de Sherbrooke, Québec

20

Constructeurs de types génériques : présentation

- CREATE DOMAIN
  - Crée un sous-type en associant une contrainte à un type.
  - Facilite la documentation, l'évolution et l'entretien des modèles logique de données.
- CREATE TYPE
  - Crée un type de base.
  - Analogue au mécanisme de classe offert par des langages tels que Objective C, C++, C#, Java, etc.

2022-01-25 SQL_01 - Types élémentaires (v401b) © 2018-2022, Μῆτις - CC BY-NC-SA 4.0
Département d'informatique, Faculté des sciences, Université de Sherbrooke, Québec

21

Voir le module BD108.

http://www.postgresql.org/docs/current/static/sql-createdomain.html
http://www.postgresql.org/docs/9.4/static/sql-createtype.html

```
CREATE DOMAIN sigle
  CHAR(6) NOT NULL
  CHECK
    (
      VALUE SIMILAR TO '[A-Z]{3}[0-9]{3}'
      AND
      VALUES IN {'IFT', 'IMN', 'IGE', 'GMQ'}
    );

CREATE DOMAIN matricule
  CHAR(8) NOT NULL
  CHECK ( VALUE SIMILAR TO '[0-9]{8}');
```

DOMAIN

2022-01-25 SQL_01 - Types élémentaires (v401b) © 2018-2022, 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.

2022-01-25 SQL_01 - Types élémentaires (v401b) © 2018-2022, Μήτις - 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*)
    [*caractéristiques_de_contrainte*]

2022-01-25 SQL_01 - Types élémentaires (v401b) © 2018-2022, Mηπς - 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 }

2022-01-25 SQL_01 - Types élémentaires (v401b) © 2018-2022, Μητις - 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);
```

2022-01-25 SQL_01 - Types élémentaires (v401b) © 2018-2022, Mηπς - 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.

2022-01-25 SQL_01 - Types élémentaires (v401b) © 2018-2022, Mṛṭṣ - 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.

2022-01-25 SQL_01 - Types élémentaires (v401b) © 2018-2022, Μήτις - CC BY-NC-SA 4.0
Département d'informatique, Faculté des sciences, Université de Sherbrooke, Québec

TYPE

Sujet pouvant être différé dans un premier temps

2022-01-25 SQL_01 - Types élémentaires (v401b) © 2018-2022, Μῆτις - 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.

2022-01-25 SQL_01 - Types élémentaires (v401b) © 2018-2022, Mηπς - 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 : le tuple

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

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

2022-01-25 SQL_01 - Types élémentaires (v401b) © 2018-2022, Mrftuς - 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

2022-01-25 SQL_01 - Types élémentaires (v401b) © 2018-2022, Μῆτις - CC BY-NC-SA 4.0
Département d'informatique, Faculté des sciences, Université de Sherbrooke, Québec

32

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

2022-01-25 SQL_01 - Types élémentaires (v401b) © 2018-2022, Μῆτις - CC BY-NC-SA 4.0
Département d'informatique, Faculté des sciences, Université de Sherbrooke, Québec

33

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>
```

2022-01-25 SQL_01 - Types élémentaires (v401b) © 2018-2022, Μῆτις - CC BY-NC-SA 4.0
Département d'informatique, Faculté des sciences, Université de Sherbrooke, Québec

34

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>

2022-01-25 SQL_01 - Types élémentaires (v401b) © 2018-2022, Μῆτις - CC BY-NC-SA 4.0
Département d'informatique, Faculté des sciences, Université de Sherbrooke, Québec

35

TYPE
Spécificité PostgreSQL – le tuple

- Comme pour ISO !

create_composite_type ::=
  CREATE TYPE *name* **AS (** [*attribute* [**,** ...]] **)**
attribute ::=
  attribute_name data_type [**COLLATE** *collation*]

2022-01-25 SQL_01 - Types élémentaires (v401b) © 2018-2022, Mητις - CC BY-NC-SA 4.0
Département d'informatique, Faculté des sciences, Université de Sherbrooke, Québec

36

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***' [, ...]])

2022-01-25 SQL_01 - Types élémentaires (v401b) © 2018-2022, 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'intervalle

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

2022-01-25 SQL_01 - Types élémentaires (v401b) © 2018-2022, 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 – 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 ]
  )
```

2022-01-25 SQL_01 - Types élémentaires (v401b) © 2018-2022, Mrftş - CC BY-NC-SA 4.0
Département d'informatique, Faculté des sciences, Université de Sherbrooke, Québec

39

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>

FCD 9075-2:2011(E) 11.50 <user-defined type definition>

13

©ISO/IEC 2010 – All rights reserved **Schema definition and manipulation 717**

FCD 9075-2:2011(E)
11.50 <user-defined type definition>

<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
Autres possibilités

- SET OF
  - constructeur d'ensembles (redondant)
- MULTISET
  - constructeur d'ensembles (redondant)
- ARRAY
  - constructeur de tableaux (à utiliser avec parcimonie)
- REF
  - pointeur (à proscrire)
- …

2022-01-25 SQL_01 - Types élémentaires (v401b) © 2018-2022, Μῆτις - CC BY-NC-SA 4.0
Département d'informatique, Faculté des sciences, Université de Sherbrooke, Québec

40

Le constructeur de type ROW est essentiel au sein de la théorie relationnelle,

Malheureusement, il souffre de plusieurs restrictions arbitraires en SQL, ce qui en limite l'utilité pratique.

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.

2022-01-25 SQL_01 - Types élémentaires (v401b) © 2018-2022, Mηπς - CC BY-NC-SA 4.0
Département d'informatique, Faculté des sciences, Université de Sherbrooke, Québec

Faire la représentation graphique au tableau

Constructeurs de types génériques : exemple

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

create domain octant_3D_pos as
  point_3D
  check (
  (value).x >= 0.0 and (value).y >= 0.0 and (value).z >= 0.0
  );

create table loc_3D (
  id integer not null,
  description text not null,
  pos point_3D not null
);
```

2022-01-25 SQL_01 - Types élémentaires (v401b) © 2018-2022, Μῆτις - CC BY-NC-SA 4.0
Département d'informatique, Faculté des sciences, Université de Sherbrooke, Québec

Constructeurs de types
Autres variantes dialectales

- Oracle
  https://docs.oracle.com/cd/E11882_01/server.112/e41084.pdf
- MS-SQL
  https://docs.microsoft.com/en-us/sql/t-sql/data-types/data-types-transact-sql
- MariaDB
  https://mariadb.com/kb/en/library/data-types/
- DB2
  https://www.ibm.com/docs/en/db2/10.5?topic=statements-create-type

2022-01-25 SQL_01 - Types élémentaires (v401b) © 2018-2022, Μῆτις - CC BY-NC-SA 4.0
Département d'informatique, Faculté des sciences, Université de Sherbrooke, Québec

Exercices

À développer en travaux dirigés, en laboratoire ou en travaux pratiques.

- booléens
- nombres
- textes
- temps

2022-01-25 SQL_01 - Types élémentaires (v401b) © 2018-2022, Μῆτις - CC BY-NC-SA 4.0
Département d'informatique, Faculté des sciences, Université de Sherbrooke, Québec

- L’éditorial
- Les références

2022-01-25 SQL_01 - Types élémentaires (v401b) © 2018-2022, Μῆτις - CC BY-NC-SA 4.0
Département d'informatique, Faculté des sciences, Université de Sherbrooke, Québec

Domaines et types
L'éditorial

- Le bon usage des domaines (types) est fondamental au développement de modèles fiables et évolutifs.
- La variabilité et l'incomplétude de la plupart des dialectes à cet égard rend cette tâche plutôt ardue à développer et difficilement transportable.

2022-01-25 SQL_01 - Types élémentaires (v401b) © 2018-2022, Μῆτις - 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 fournisseurs de SGBD adhèrent strictement à la norme plutôt que de pousser des dialectes.

2022-01-25 SQL_01 - Types élémentaires (v401b) © 2018-2022, Mητις - CC BY-NC-SA 4.0
Département d'informatique, Faculté des sciences, Université de Sherbrooke, Québec

Faire la représentation graphique au tableau

Références

- [Loney2008]
  Loney, Kevin ;
  Oracle Database 11g: The Complete Reference.
  Oracle Press/McGraw-Hill/Osborne, 2008.
  ISBN 978-0071598750.
- [Date2012]
  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 de PostgreSQL (en français)
  - https://doc.postgresqlfr.org
- Le site de PostgreSQL (en anglais)
  - https://www.postgresql.org
- Le site d'Oracle (en anglais)
  - https://docs.oracle.com/en/database/oracle/oracle-database/21/development.html

2022-01-25 SQL_01 - Types élémentaires (v401b) © 2018-2022, Μῆτις - CC BY-NC-SA 4.0
Département d'informatique, Faculté des sciences, Université de Sherbrooke, Québec