Bases de données
SQL

Types élémentaires et prédéfinis

SQL_01
v401b

2022-01-25

UNIVERSITÉ DE SHERBROOKE

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

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*

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

Types prédéfinis ISO

Types prédéfinis ISO (présents depuis ISO 9075:2003)

Les nombres

- SMALLINT
- INTEGER
- BIGINT

- NUMERIC (p,s)
- DECIMAL (p,s)

- FLOAT (p)
- REAL
- DOUBLE PRECISION

Les autres

- BOOLEAN

- CHARACTER (n)
  CHAR (n)
- CHARACTER VARYING (n)
  VARCHAR (n)

- DATE
- TIME (p)
- TIMESTAMP (p)
- INTERVAL (p)

Types prédéfinis ISO : type booléen

- BOOLEAN (FALSE, TRUE, UNKNOWN)

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

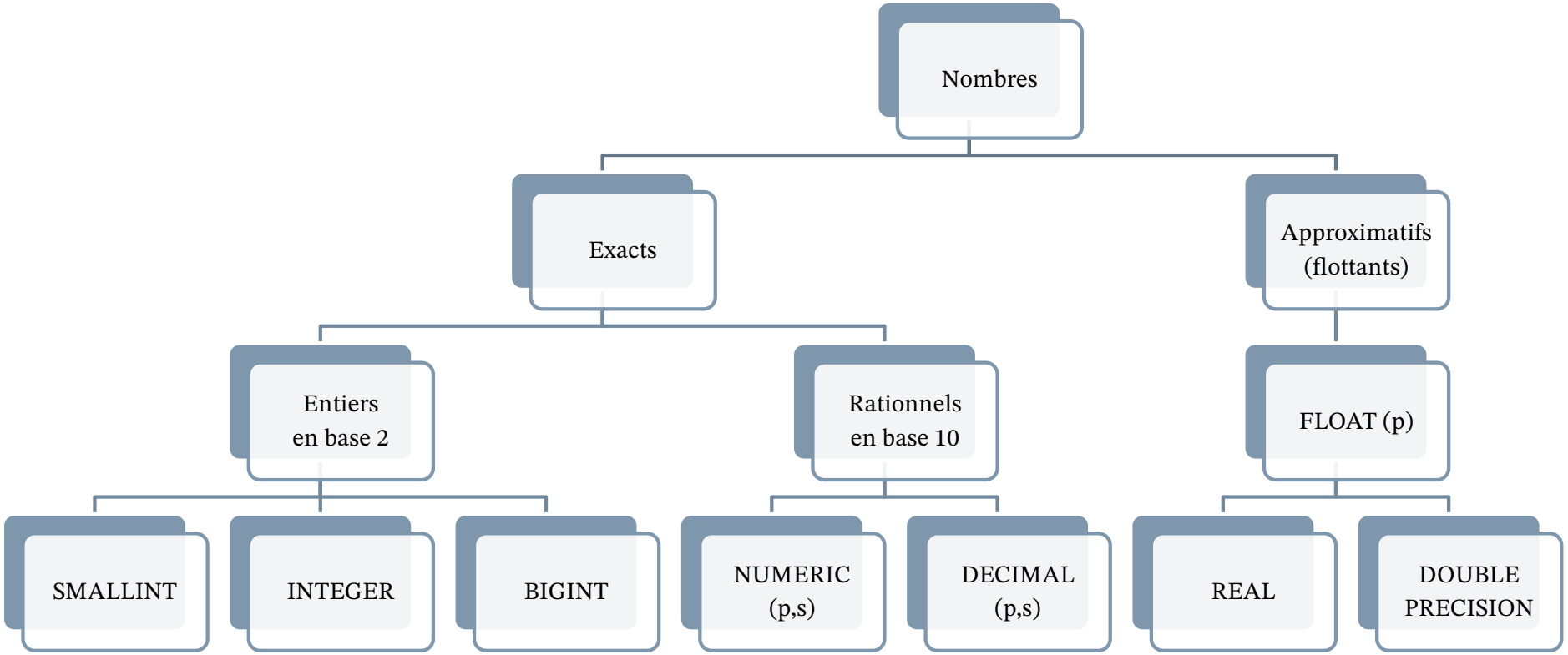
- Dans les deux cas, il **faut** de suffixer une limite entre parenthèses :
  - exacte dans le cas de CHAR
  - maximale dans le cas de VARCHAR

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

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

Types prédéfinis ISO : types numériques



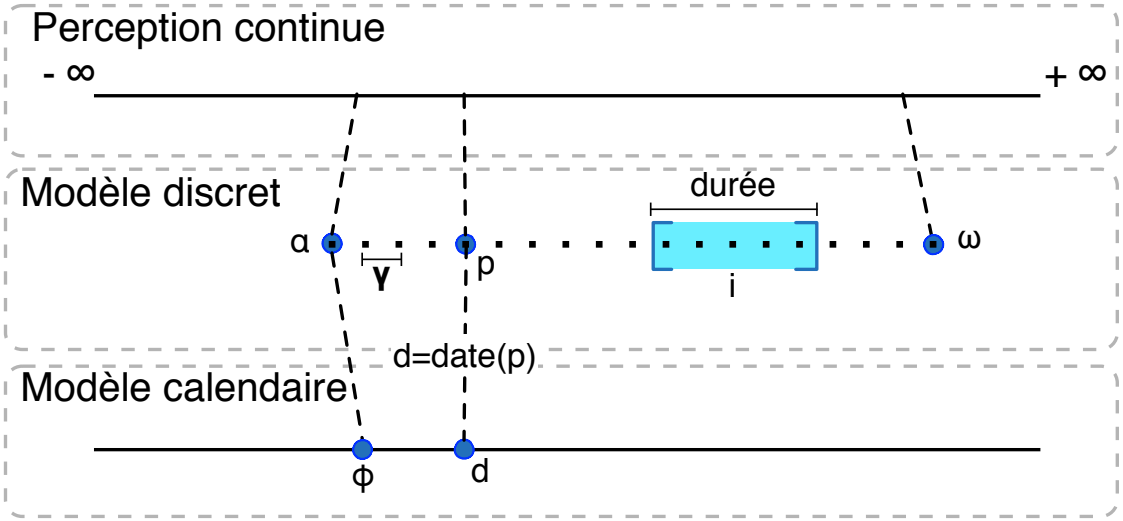
Types prédéfinis ISO
types temporels

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



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

durée(i) = card(i) * γ

date(α) = ϕ
$p_i < p_j \Rightarrow date(p_i) \leq date(p_j)$

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

Types prédéfinis PostgreSQL

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

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

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

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

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

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}.

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.

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

Types de stockage : grands objets

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

Types non utilisés en cours

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

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.

DOMAIN

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*)
[*caractéristiques_de_contrainte*]

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.

TYPE

Sujet pouvant être différé dans un premier temps

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 : le tuple

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 – le tuple

- Comme pour ISO !

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

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

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
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)
- …

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.

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
);
```

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

Exercices

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

- booléens
- nombres
- textes
- temps

- L’éditorial
- Les références

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.

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.

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