Bases de données
SQL

Types élémentaires et prédéfinis

SQL_01
v411b
2025-01-20

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

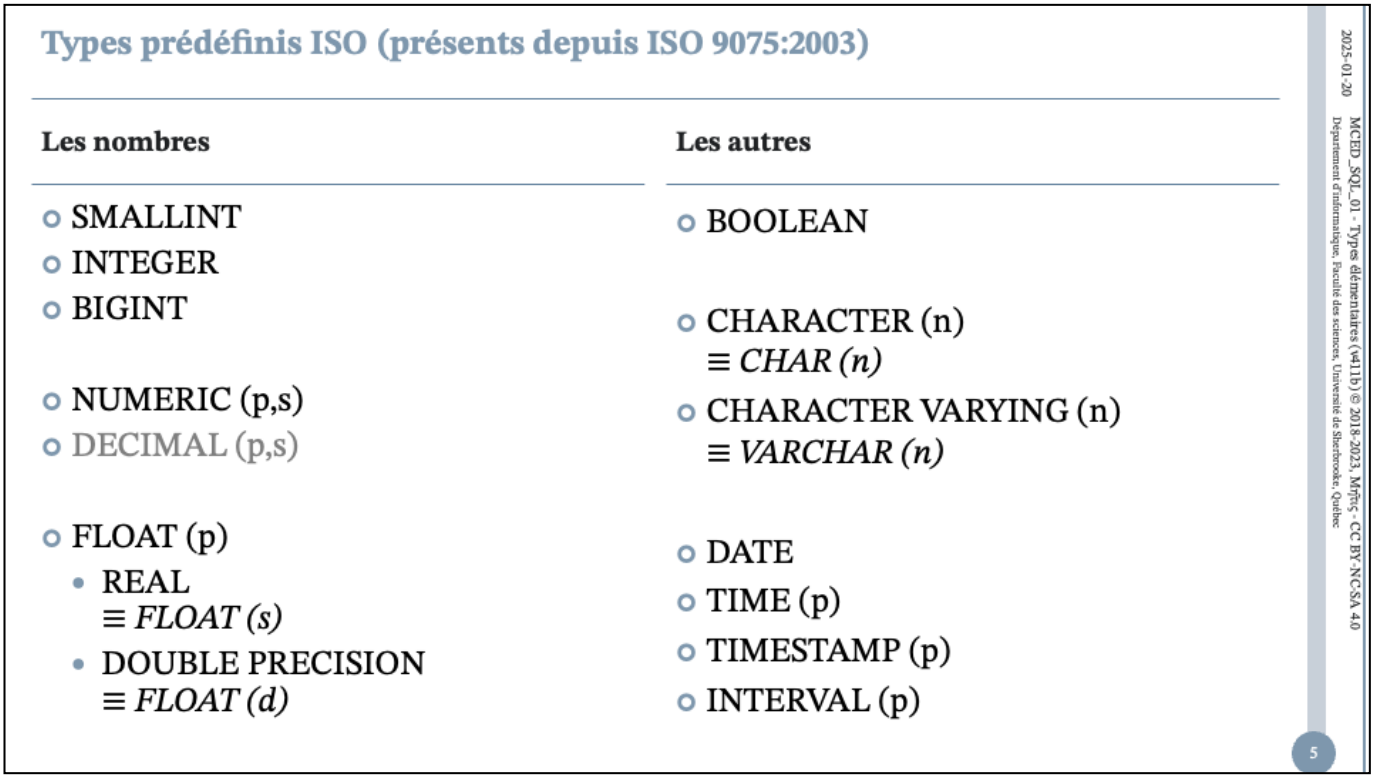
- Types prédéfinis en SQL
  - Type booléen
  - Types textuels
  - Types numériques
  - Types temporels
- Types prédéfinis en PostgreSQL
  - Type booléen
  - Types textuels
  - Types numériques
  - Types temporels

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 ISO

- Revue des principaux types prescrits par la norme ISO 9075:2023.



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

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>.

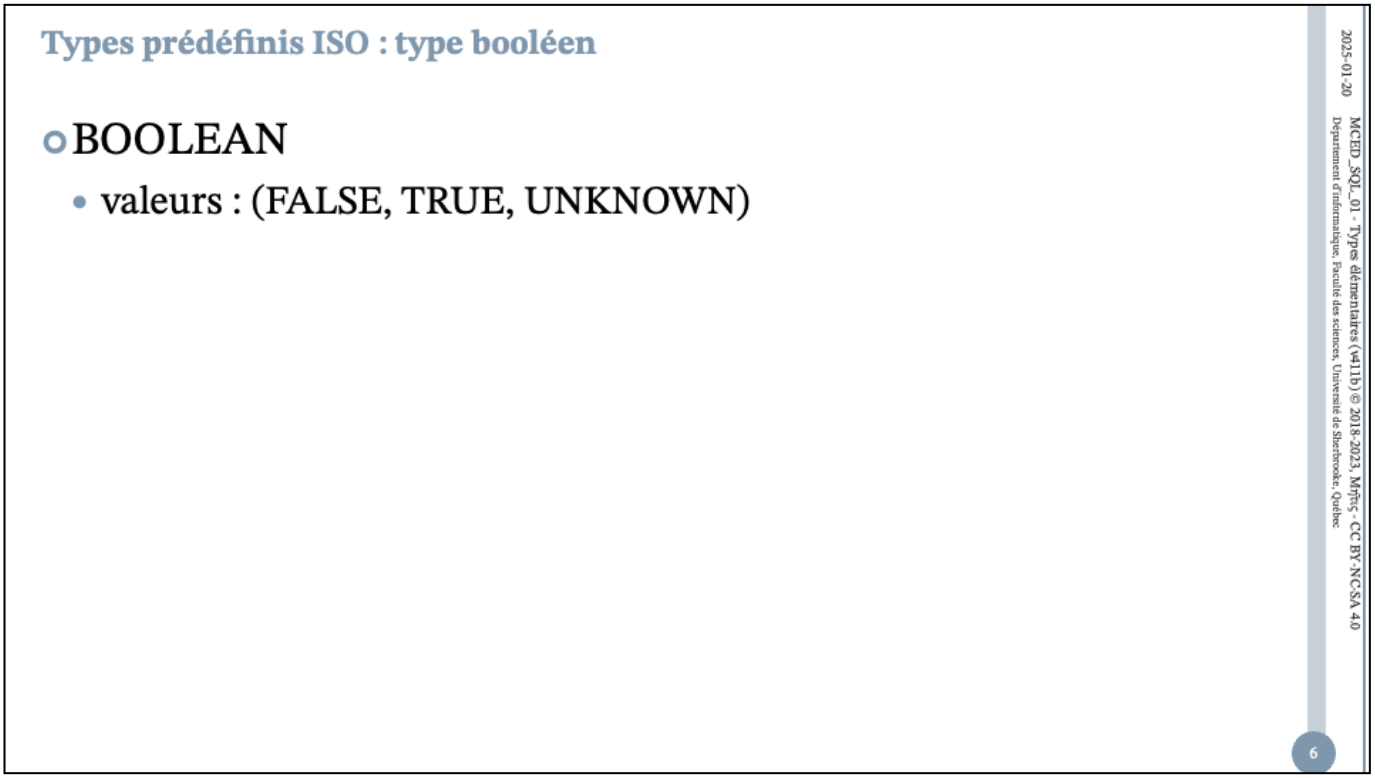
22) DECIMAL specifies the data type exact numeric, with the decimal scale specified by the <scale> and the implementation-defined decimal precision equal to or greater than the value of the specified <precision>.

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.



La valeur unknown n'est pas reconnue par tous les dialectes.

Pourfois, il faut utiliser NULL, CAST(NULL AS BOOLEAN), voire null::boolean.

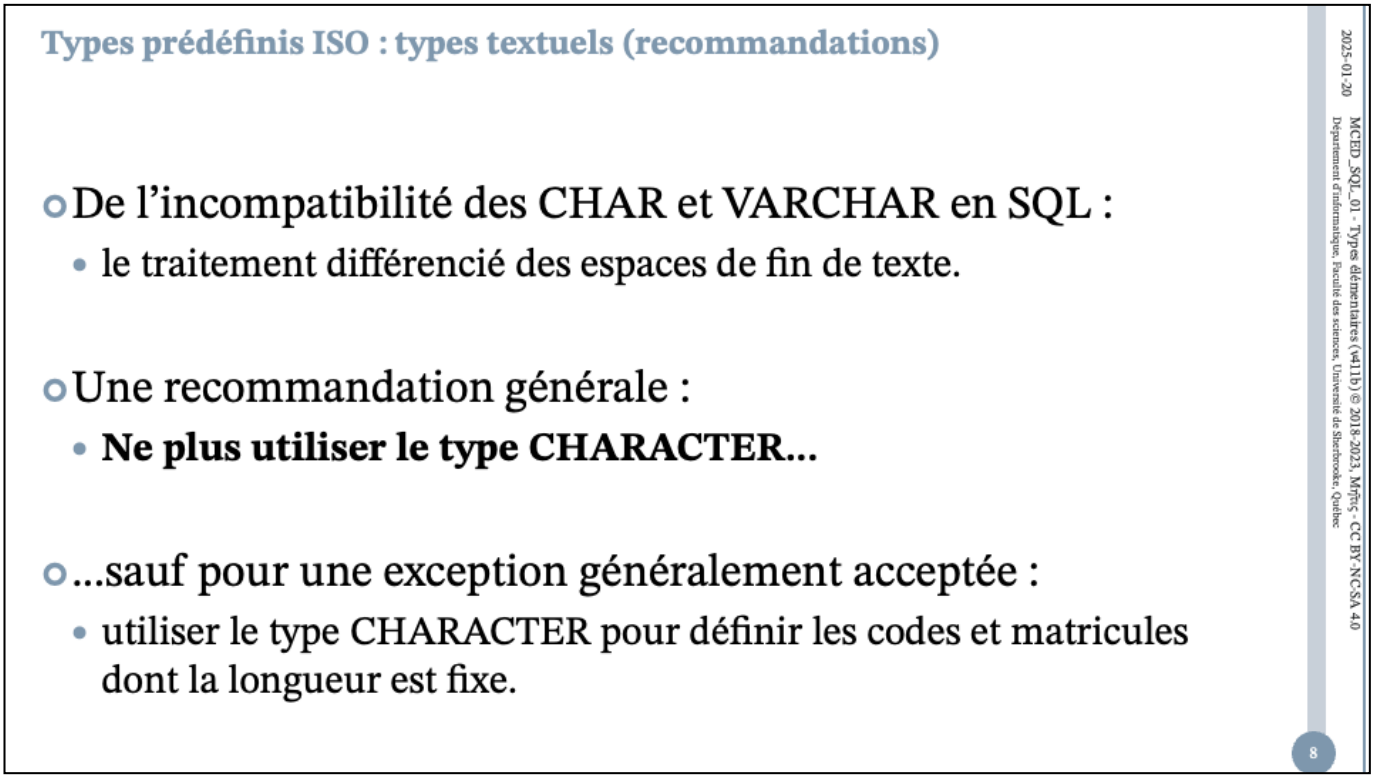
Types prédéfinis ISO : types textuels

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

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

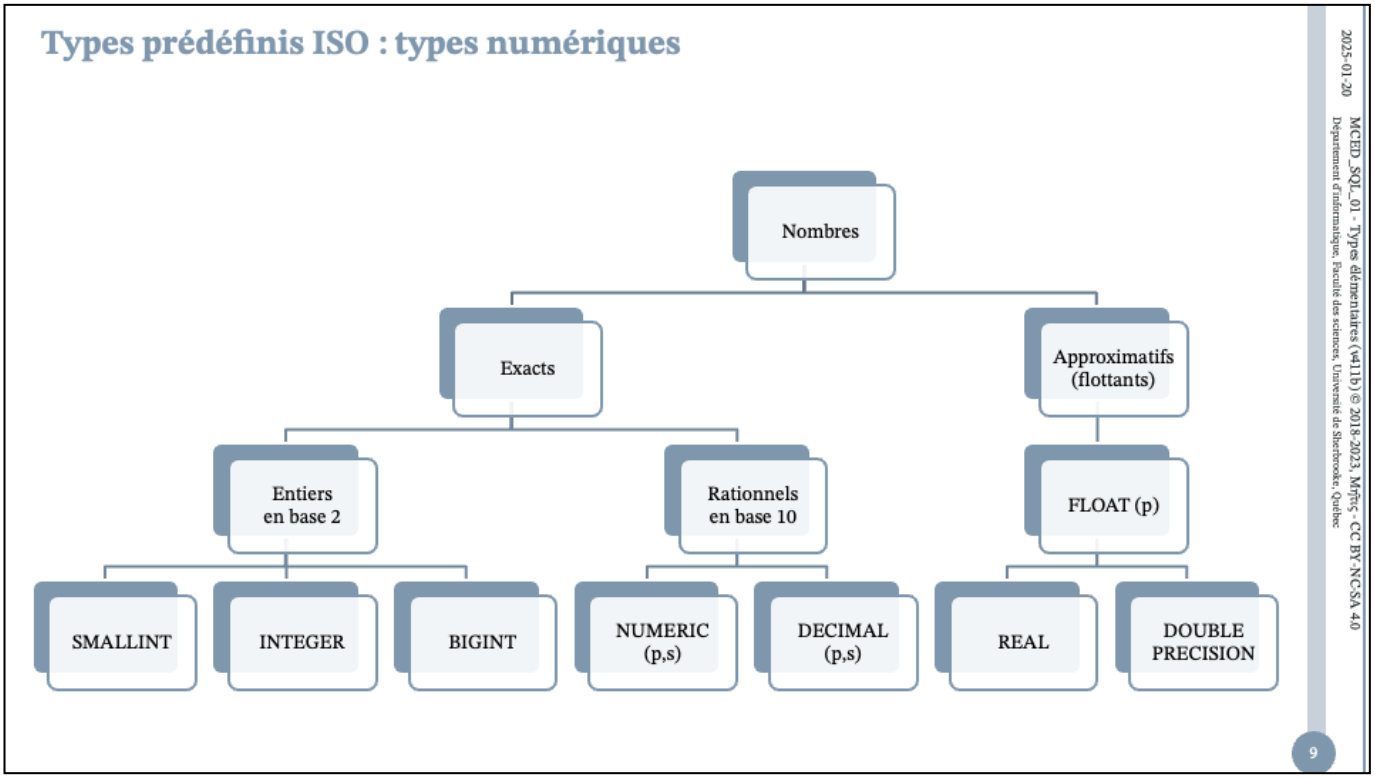
- Dans les deux cas, il **faut** suffixer une limite entre parenthèses :
  - *exacte* dans le cas de CHAR
  - *maximale* dans le cas de VARCHAR

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



En fait, le type CHAR est une « optimisation » tirant profit du fait que le texte est toujours de la même longueur.

Comme toute optimisation, il ne devrait être utilisé qu'en dernier recours, lorsque son utilisation à un impact positif significatif compensant pour la perte de généralité (et donc d'évolutivité).



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

Approximatifs

SQL prescrit le respect du standard IEEE Standard 754 (Binary Floating-Point Arithmetic)

REAL et DOUBLE PRECISION ne sont que des cas particuliers de la notation générale FLOAT (p)
où p représente le nombre minimal de bit de la mantisse :
* REAL == FLOAT (25)
* DOUBLE PRECISION == FLOAT(53)

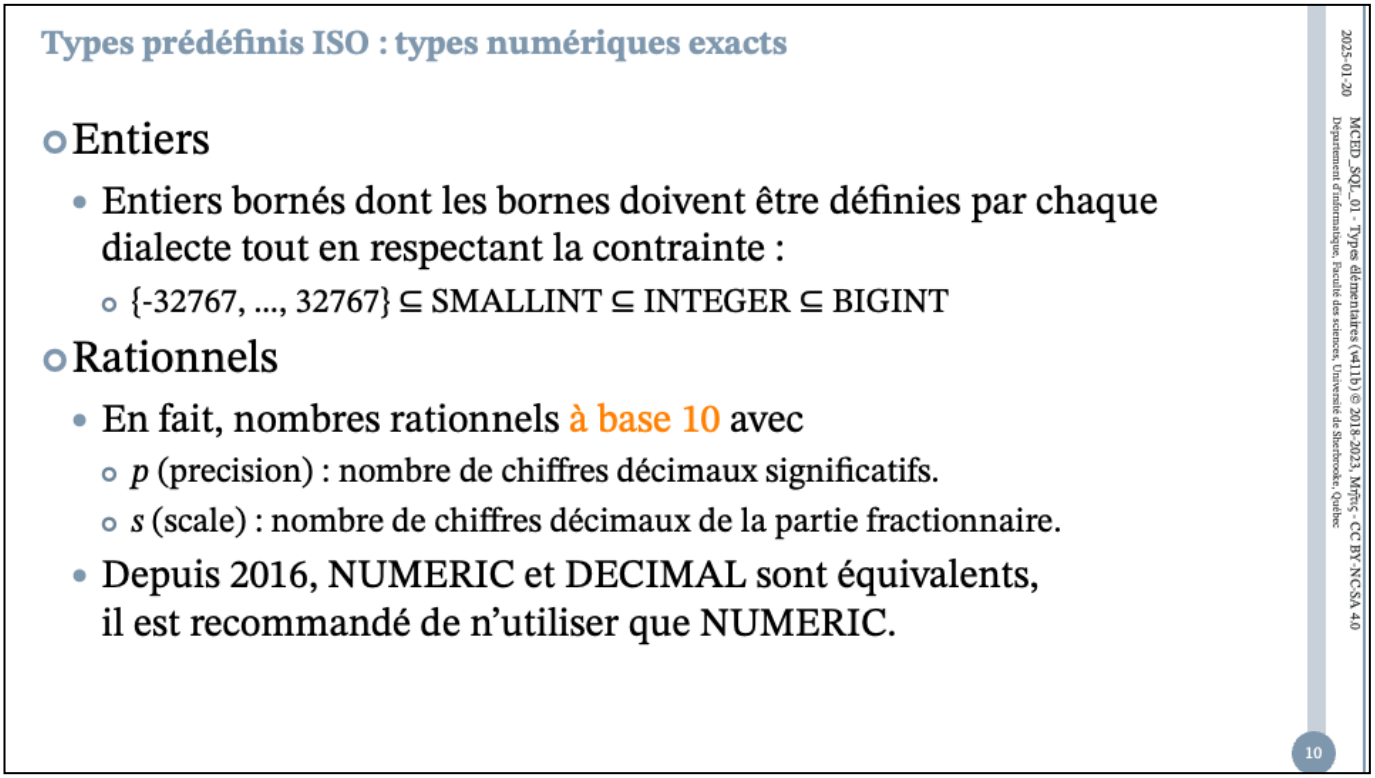
Rationnels

En fait, nombres rationnels à base 10.
Désormais (depuis 2016) NUMERIC et DECIMAL sont équivalents.
p(precision) :
s(cope) :

Entiers

Entiers bornés, les bornes devant être définies par chaque dialecte tout en respectant la contrainte :

{-32767, ..., 32767} ⊆ SMALLINT ⊆ INTEGER ⊆ BIGINT



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.

Une différence existait autrefois entre numeric et decimal : numeric devait fournir EXACTEMENT la précision demandée, decimal AU MOINS la précision demandée.

Typiquement (et notamment dans le dialecte PostgreSQL) :

a) Entiers (smallint sur 16 bits, integer sur 32 bits, bigint sur 64 bits).

b) Rationnels fixes décimaux (p et s indiquent le nombre de chiffres, p pour la précision et s pour l'échelle de 10^-s ; s peut être négatif... pour indiquer une échelle supérieure à 1, par exemple -2 pour une échelle de 100 = 10^-(-2)).

Types prédéfinis ISO : types numériques approximatifs

- Approximatifs
  - SQL prescrit le standard
    IEEE Standard 754 Binary Floating-Point Arithmetic ;
  - *p* (précision) représente le nombre minimal de bits de la mantisse ;
  - Exemples
    - REAL ≡ FLOAT (25)
    - DOUBLE PRECISION ≡ FLOAT(53)

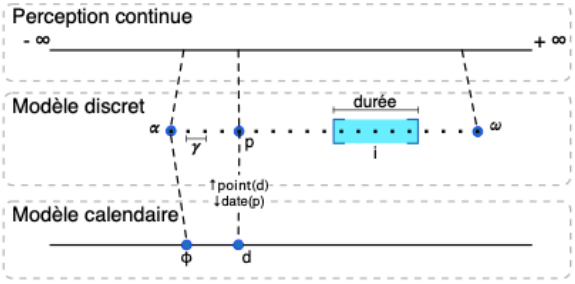
Types prédéfinis ISO
types temporels

Le nécessaire

- TIMESTAMP :
  un point sur l'axe du temps (p) ;
- INTERVAL :
  une durée.

L'utile

- DATE :
  une référence à un jour d'un calendrier ;
- TIME :
  une référence à un moment de la journée.



α : point initial (minimum)
ω : point final (maximum)
γ : distance entre deux points consécutifs = 1 / Γ
Γ : nombre de points dans une seconde

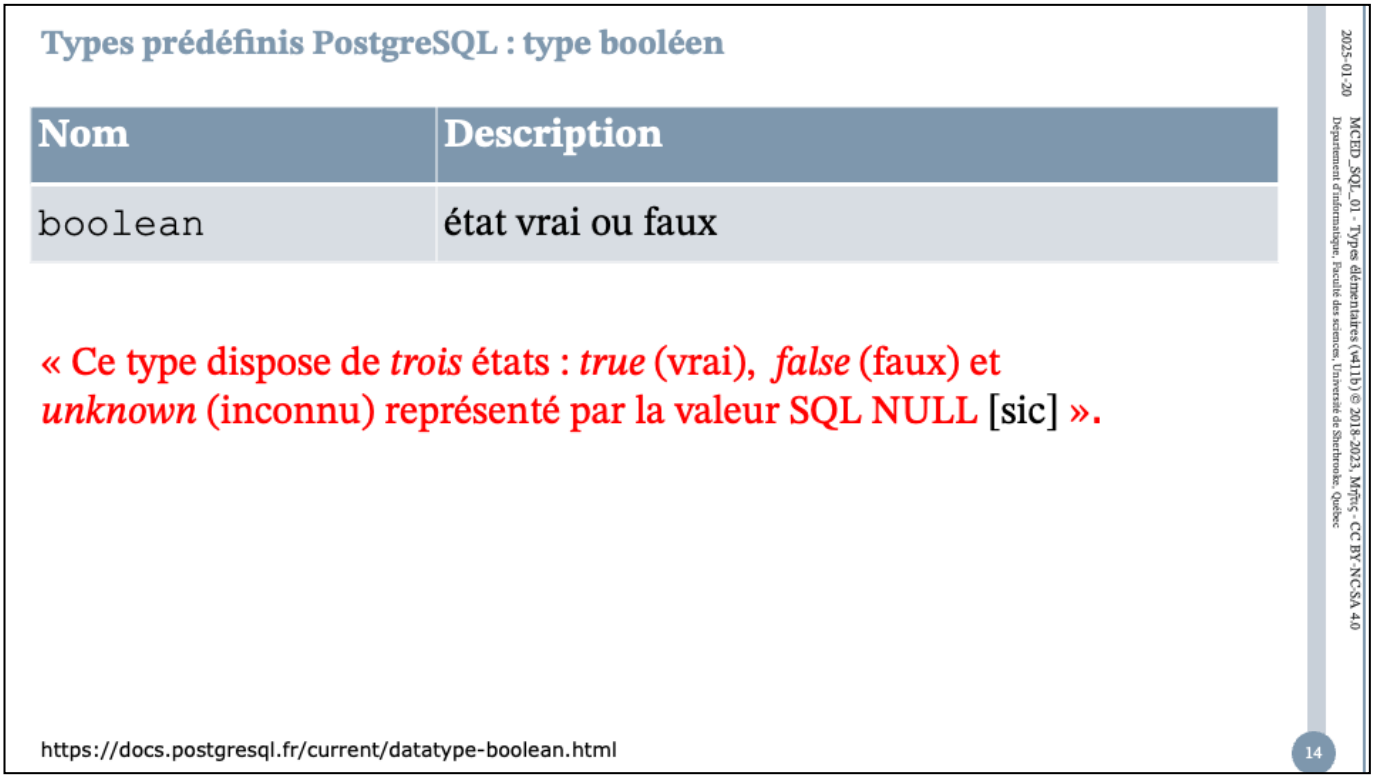
i : intervalle quelconque
p, p_a, p_b : points quelconques
d, d_a, d_b : dates quelconques

$durée(i) = (card(i)-1) * \gamma$

$date(\alpha) = \phi$
$point(\phi) = \alpha$
$p_a < p_b \Rightarrow date(p_a) \leq date(p_b)$, *mais point(date(p)) = p n'est pas garanti*
$d_a < d_b \Rightarrow point(d_a) \leq point(d_b)$, *mais date(point(d)) = d n'est pas garanti*

Types prédéfinis PostgreSQL

- PostgreSQL est en général conforme à la norme
  - *ISO 9075:2016*
- Les précisions requises par la norme sont fournies.
- Plusieurs « extensions » sont également offertes, mais seules quelques-unes seront couvertes ici.



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'

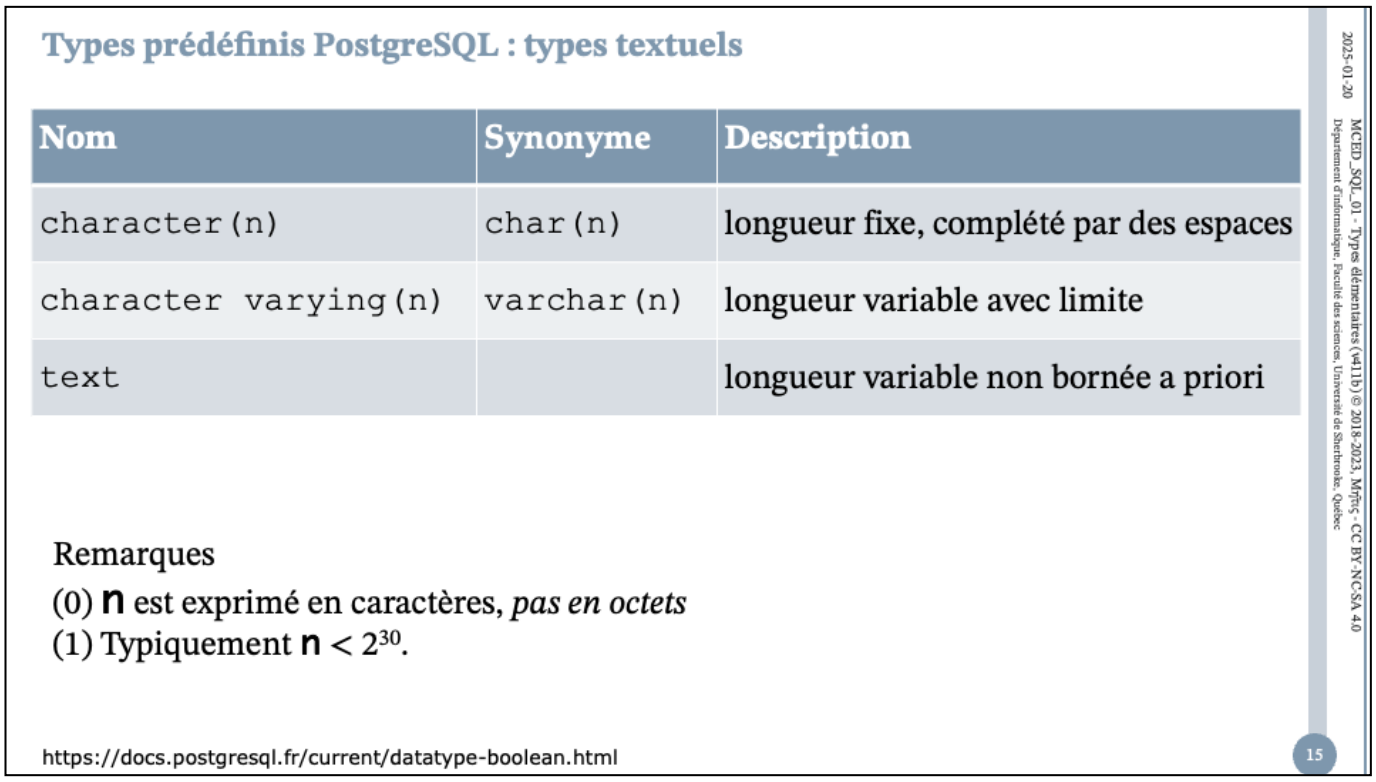
'n'

'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(n)`	`char(n)`	longueur fixe, complété par des espaces
`character varying(n)`	`varchar(n)`	longueur variable avec limite
`text`		longueur variable non bornée a priori

Remarques
(0) **n** est exprimé en caractères, *pas en octets*
(1) Typiquement **n** < 2^{30}.

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

15

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 habituel	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

16

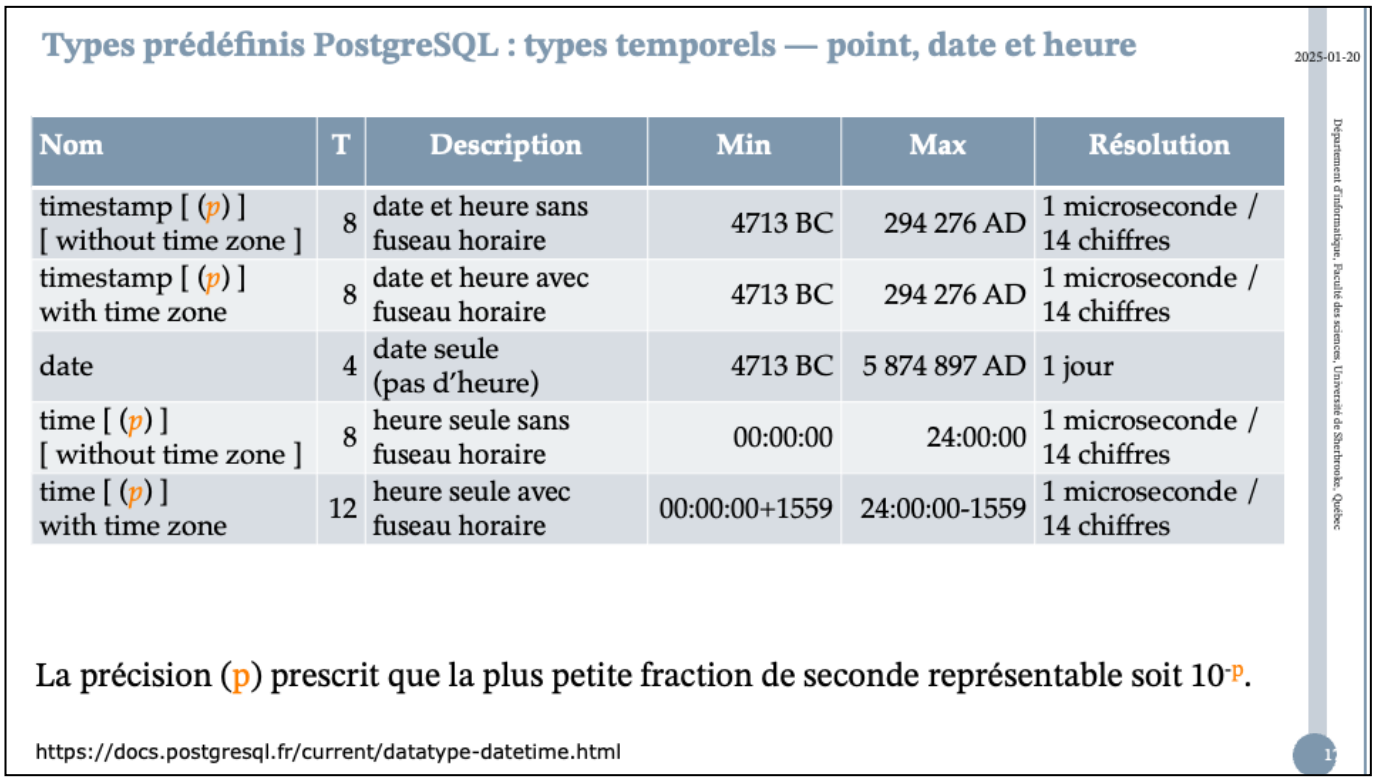
La colonne T représente la taille en octets; la mention v signifie que c'est variable en fonction de la « longueur » de la valeur.

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.

REAL est en fait défini par FLOAT (25)

DOUBLE PRECISION est en fait défini par FLOAT (53)



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

2025-01-20

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

Les 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.

Exemples :

date '2015-09-01'

time '10:30'

timestamp '2015-09-01 10:30:00.123456'

L'indication WITH TIME ZONE indique que le temps est représenté relativement aux fuseaux horaires UTC.

AVERTISSEMENT :

L'affirmation suivante N'est PAS vérifiée pour tous les dialectes SQL.

«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 exprimée dans le fuseau horaire du système référence. Elle est alors convertie implicitement en UTC en utilisant le décalage approprié. »

PAR AILLEURS :

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 années	178 000 000 années	1 microseconde / 14 chiffres

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

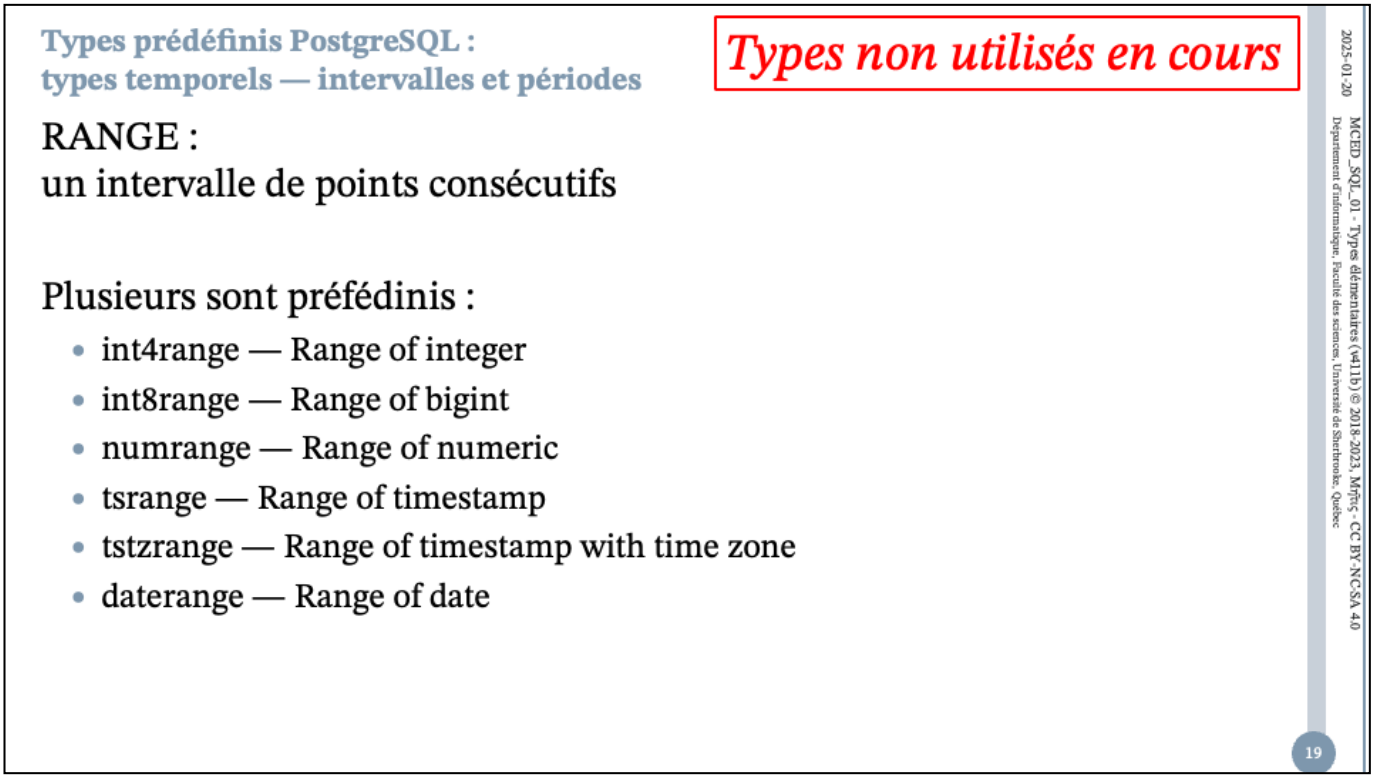
- YEAR
- MONTH
- DAY
- HOUR
- MINUTE
- SECOND
- YEAR TO MONTH
- DAY TO HOUR
- DAY TO MINUTE
- DAY TO SECOND
- HOUR TO MINUTE
- HOUR TO SECOND
- MINUTE TO SECOND

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}.

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.



Un identifiant plus approprié aurait été INTERVAL, mais il est déajà utilisé pour désigner la durée!!!

Types prédéfinis de stockage

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

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

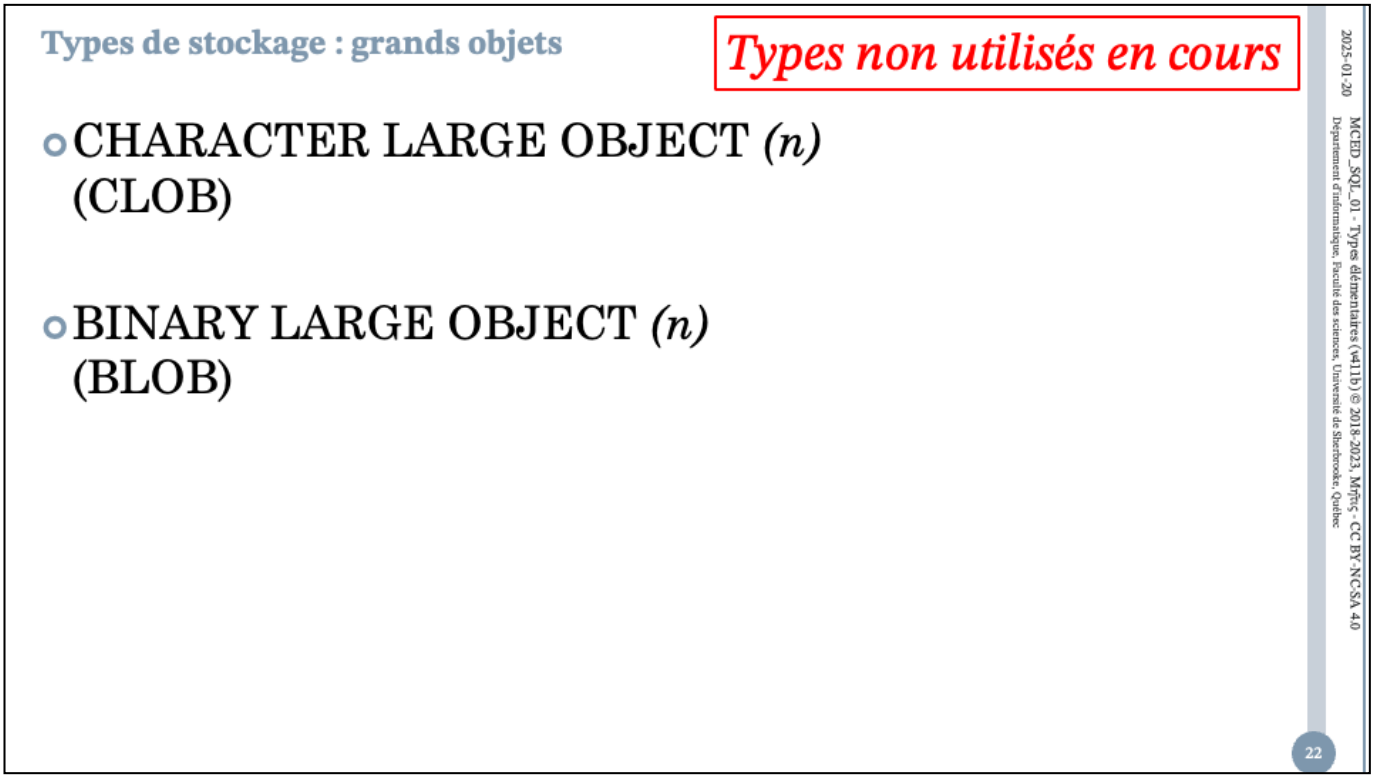
Leur usage prépondérant est le transport de données chiffrées.

Sujet pouvant être différé dans un premier temps

Types de stockage : petits objets

Types non utilisés en cours

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



La notion de LARGE varie beaucoup d'un dialecte à un autre :

* Oracle, 4 ko et plus ;
* PostgreSQL, 4 Go et plus.

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

- booléens
- nombres
- textes
- temps

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

Conclusion

- L'éditorial
- Les références

L'éditorial
Domaines et types

- Le bon usage des types est fondamental au développement de modèles fiables et évolutifs.
- À cet égard, le langage SQL est probablement un des plus riches parmi les langages couramment utilisés.
- Par contre, la variabilité et l'incomplétude de la plupart des dialectes à cet égard rend difficilement transportable (et donc généralisable) l'utilisation de certains des mécanismes de typage.

L'éditorial
SQL ISO

- *ISO or not ISO, that's the question !*

- 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
  - plus particulièrement le chapitre 8 : https://doc.postgresql.fr/15/datatype.html
- 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