Bases de données
SQL

Opérateurs pour les nuls

SQL_08
v420a

2025-09-02

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

Plan

- NULL : marqueur ou valeur ?
- Impacts sur la logique
- Impacts sur les opérateurs relationnels
- Impacts sur les clés
- Opérateurs spéciaux
- Jointures externes

NULL : marqueur ou valeur ?

- NULL considéré comme marqueur
- NULL considéré comme valeur

NULL considéré comme un marqueur d'attribut (1/3)

- NULL marque l'absence de valeur, ainsi
  - INSERT INTO R (a, b) VALUES (12, NULL)

  a pour effet
  - d'affecter la valeur 12 à l'attribut a ;
  - de marquer l'attribut b comme étant NULL (sans valeur associée) ;
  - si b n'est pas annulable, c'est une erreur.

NULL considéré comme un marqueur d'attribut (2/3)

- Soit x un attribut quelconque, on **ne peut pas** évaluer
  - $x = \text{NULL}$

  on doit évaluer le statut du marqueur ainsi
  - x IS NULL

NULL considéré comme un marqueur d'attribut (3/3)

- Soit *a* et *b* deux attributs dont au moins un est marqué NULL,
  - $a = b$ est inconnu
  - $a <> b$ est inconnu aussi

- En particulier, si *a* est marqué NULL
  - $a = a$ est inconnu
  - $a <> a$ est inconnu aussi

NULL considéré comme une valeur (1/2)

- En général, toute évaluation d'expression nécessitant l'évaluation d'un attribut marqué NULL entraine « l'annulabilité de l'expression ».
- Or comment une expression pourrait-elle être « nulle » si NULL est un marqueur d'attribut (*seulement*) ?
- Conséquemment, **il y aura aussi** des valeurs nulles afin de permettre l'évaluation des expressions.
- Ceci s'applique donc aux expressions logiques aussi.

NULL considéré comme une valeur (2/2)

- Ainsi, la *valeur* d'un prédicat dont un des termes a pour valeur NULL est inconnue (UNKNOWN).

- La logique de SQL est donc trivaluée :
  FALSE, TRUE et UNKNOWN
  et les opérateurs logiques usuels doivent donc été « étendus » en conséquence, mais de façon différente de celle utilisée pour les valeurs non logiques (un terme UNKNOWN ne forcera l'expression à être nécessairement UNKNOWN également).

- Finalement, l'évaluation des opérateurs IS NUL, IS NOT NULL, NULLIF et COALESCE relèveront de règles différentes encore.

Le problème de l'égalité

- Une valeur non nulle est-elle égale à une valeur nulle ?
- Deux valeurs nulles sont-elles égales ?

- À considérer (liste partielle)
  - les expressions arithmétiques
  - les calculs statistiques
  - les opérateurs de jointure, d'union et de différence

Impacts sur la logique

- Opérateurs (« tables de vérité »)
- Assertions et contraintes

Logique non classique utilisée par SQL

OR	true	false	unknown
true	true	true	true
false	true	false	unknown
unknown	true	unknown	unknown

AND	true	false	unknown
true	true	false	unknown
false	false	false	false
unknown	unknown	false	unknown

P	NOT P
true	fase
false	true
unknown	unknown

IS	true	false	unknown
true	true	false	false
false	false	true	false
unknown	false	false	true

Assertions et contraintes
L'incohérence du système logique de SQL

- Que se passe-t-il quand la condition d'une contrainte (CHECK) est UNKNOWN ?
  - Elle est réputée satisfaite !
- Que se passe-t-il quand la condition d'une restriction (WHERE) est UNKNOWN ?
  - Elle est réputée non satisfaite !!!
- Le système logique utilisé par SQL est donc incohérent!

Conséquences

- L'égalité est également incohérente en présence de NULL et UNKNOWN.
- Les opérateurs l'utilisant implicitement (dont l'union, l'intersection, la différence, la restriction et la jointure) peuvent avoir des comportements incohérents.
- Conséquemment les instructions DELETE, UPDATE et SELECT peuvent également produire des résultats incohérents comme nous le verrons dans les prochains modules.

Impacts sur opérateurs relationnels

- Sauf la projection et le renommage, tous les opérateurs sont touchés et « étendus » :
  - par l'extension de la logique (UNKNOWN -> FALSE)
    - WHERE
  - par l'extension de l'égalité (UNKNOWN -> FALSE...
    sauf dans le cas où les deux valeurs sont nulles -> TRUE)
    - JOIN,
    - UNION
    - INTERSECT
    - EXCEPT

Impacts sur les clés

- Clés candidates
  - primaires
  - secondaires
- Clés référentielles

Impact sur les clés candidates primaires
PRIMARY KEY

- Aucun impact, les attributs d'une clé candidate primaire ne peuvent être annulables
  - SQL impose d'ailleurs automatiquement la contrainte **NOT NULL** aux attributs participant à une clé candidate primaire.

Impact sur les clés candidates secondaires
UNIQUE

- Bien que le comportement de l'égalité puisse être modulé au moment de la déclaration que de la contrainte

  cléSecondaire ::=

  UNIQUE [NULLS [NOT] DISTINCT] (*listeNomsColonne* **)**

- Il est fortement recommandé de ne pas permettre la participation d'attributs annulables à une clé candidate, fut-elle secondaire.

Impact sur les clés référentielles

- Bien que le comportement de l'égalité puisse être modulé au moment de la déclaration que de la contriante

  cléRéférentielle ::=

  FOREIGN KEY (*listeNomsColonne* **)**
  REFERENCES *nomTable* [**(** *listeNomsColonne* **)**]
  [**MATCH** { **SIMPLE** | **PARTIAL** | **FULL** }]
  [**ON UPDATE** *action*]
  [**ON DELETE** *action*]

  action ::=

  CASCADE | **SET NULL** | **SET DEFAULT** | **NO ACTION**

- Il est fortement recommandé de ne pas permettre la participation d'attributs annulables à une clé référentielle.

Opérateurs spéciaux

- COALESCE
- NULLIF
- Jointures externes

Formes abrégées du CASE, l'opérateur COALESCE

- COALESCE (*V*1, *V*2) est équivalent à :
  - ```
    CASE
    WHEN V1 IS NOT NULL THEN V1
    WHEN V2 IS NOT NULL THEN V2
    ELSE NULL
    END
    ```
- COALESCE (*V*1, *V*2, ..., *Vn*), pour $n \geq 3$, est équivalent à :
  - ```
    CASE
    WHEN V1 IS NOT NULL THEN V1
    ELSE COALESCE (V2, ..., Vn)
    END
    ```

Opérateur COALESCE
Autrement dit…

- COALESCE (x_1, x_2, …, x_n)
  - première(*) expression non nulle parmi x_1, x_2, …, x_n ;
  - si toutes les expressions sont nulles, NULL.

  - (*) de gauche à droite
  - (*) dont l'indice est le plus petit

Formes abrégées du CASE - NULLIF

- NULLIF (*V*1, *V*2) est équivalent à :
  - `CASE`
    `WHEN` *V*1`=`*V*2 `THEN NULL`
    `ELSE` *V*1
    `END`
- Mais de quelle égalité s'agit-il ?
  - La clause `WHEN` considère qu'un résultat `UNKNOWN` est **faux**

Opérateur NULLIF
Autrement dit…

- NULLIF (x_1, x_2)
  - si x_1 est nul alors NULL
  - sinon si x_2 est NULL alors x_1
  - sinon si $x_1 = x_2$ alors NULL
  - sinon x_1

Jointures externes

jointure_externe ::=
  jExterne JOIN *denotationTable* [[AS] *alias*] *qualificationJ*
jExterne ::=
  LEFT [OUTER] | RIGHT [OUTER] | FULL [OUTER]

- Une jointure externe permet de ne pas « perdre » les données des tuples sans correspondant en affectant des NULL aux attributs de valeur inconnue.
- Le mot OUTER est superfétatoire (sic).
- Le mot AS permet de (re)nommer la *dénotationTable* en même temps.

Le Langage SQL
Illustration jointure externe

jointure-gauche (LEFT JOIN)

A	B
a1	b1
a2	b1
a3	b3
a4	b4

LEFT JOIN

B	C
b1	c1
b2	c2
b3	c3
b5	c4

=

A	B	C
a1	b1	c1
a2	b1	c1
a3	b3	c3
a4	b4	NULL

jointure-complète (FULL JOIN)

A	B
a1	b1
a2	b1
a3	b3
a4	b4

FULL JOIN

B	C
b1	c1
b2	c2
b3	c3
b5	c4

=

A	B	C
a1	b1	c1
a2	b1	c1
NULL	b2	c2
a3	b3	c3
a4	b4	NULL
NULL	b5	c4

jointure-droite (RIGHT JOIN)

A	B
a1	b1
a2	b1
a3	b3
a4	b4

RIGHT JOIN

B	C
b1	c1
b2	c2
b3	c3
b5	c4

=

A	B	C
a1	b1	c1
a2	b1	c1
NULL	b2	c2
a3	b3	c3
NULL	b5	c4

Le langage SQL
Jointures externes simplifiées

```
SELECT C.nom, T.tel
FROM
  C RIGHT JOIN T ON C.id = T.id
```

est équivalent à :

```
SELECT C.nom, T.tel
FROM
  T LEFT JOIN C ON C.id = T.id
```

```
SELECT C.nom, T.tel
FROM
  C FULL JOIN T ON C.id = T.id
```

est équivalent à :

```
SELECT C.nom, T.tel
FROM
  C RIGHT JOIN T ON C.id = T.id
UNION
SELECT C.nom, T.tel
FROM
  C LEFT JOIN T  ON C.id = T.id
```

Conclusion

- Comment rendre cohérente
  - le traitement de l'égalité en présence de valeurs nulles et
  - le traitement de la tautologie et de l'inférence en logique ?

- Réponse
  - Aucune solution satisfaisante n'est présentement connue avec une logique à trois valeurs.

- Conclusion
  - Ne serait-il pas plus sage de s'en tenir à une logique à deux valeurs ?

Conclusion (suite)

- Pour cette raison, plusieurs auteurs (et non des moindres) préconisent d'utiliser uniquement une logique à deux valeurs (celles de Boole) au sein du modèle relationnel.
- En pratique (en SQL), cela signifie donc
  - de déclarer tout attribut comme étant non annulable (NOT NULL)
  - de prendre en compte la possibilité d'absence d'une donnée par la modélisation (décompostion de porjection-jointure ou de restriction-union)

Références

- Elmasri et Navathe (4e ed.), chapitre 7
- Elmasri et Navathe (6e ed.), chapitre 4
- [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.
- La documentation de PostgreSQL (en français)
  - https://docs.postgresql.fr
- La documentation de MariaDB (en anglais)
  - https://mariadb.com/kb/en/library/documentation/
- La documentation d'Oracle (en anglais)
  - http://docs.oracle.com/cd/E11882_01/index.htm