Bases de Données 2
#1 - SQL

Matthieu Nicolas
Polytech S5 - II
Slides réalisées à partir de celles de Claude Godart et Malika Smaïl

Plan

- Interrogation d’une BD avec SQL - avancé

Interrogation d’une BD avec SQL - avancé

Base de Données 2
#2

Opérateurs manquants

- On a vu dans le cours précédent comment étaient convertis en SQL plusieurs opérateurs de l'algèbre relationnel
  - PROJECT, RESTRICT, JOIN, PRODUCT
- Mais il nous en manque encore quelques uns
  - UNION, INTERSECT, MINUS, DIV
- On va essayer de combler nos lacunes

Exemple

- On re-considère les relations suivantes pour les exemples suivants :

Produits(<u>prod_id</u>, nom, pu)

prod_id	nom	pu
1	A3	10.0
2	crayon	9
3	stylo	15
4	A4	10.0

Depots(<u>depot_id</u>, adr, volume)

depot_id	adr	volume
1	Nancy	100
2	Laxou	200
3	Vandoeuvre	115
4	Nancy	220
5	Nancy	1000

Stock(<u>prod_id, depot_id</u>, qte)

prod_id	depot_id	qte
1	1	0
3	2	9
1	3	15
2	4	20
3	5	0
1	4	5
2	5	2
3	3	30
1	2	10

Union

- Permet de calculer l'union entre 2 relations équivalentes

SELECT <liste d'attributs projetés>

FROM <liste de relations>

WHERE <liste de critères de sélection et de jointure>

UNION

SELECT …

FROM …

WHERE …

Exemple Union

- Dépôts ayant un volume supérieur à 500 ou se trouvant à Laxou

```
SELECT *
FROM Depots
WHERE volume > 500
UNION
SELECT depot_id, adr, volume
FROM Depots
WHERE adr = “Laxou”
```

Res(depot_id, adr, volume)

depot_id	adr	volume
2	Laxou	200
5	Nancy	1000

Intersection - 1

- Permet de calculer l'intersection entre 2 relations équivalentes

- S'utilise normalement de la façon suivante en SQL :

SELECT <liste d'attributs projetés>

FROM <liste de relations>

WHERE <liste de critères de sélection…>

INTERSECT

SELECT …

FROM …

WHERE …

Intersection - 2

- Ce n’est pas le cas en MySQL…
- … à la place, utilise l’opérateur IN pour faire l’intersection sur un ensemble d’attributs

Exemple Intersection

- Produits stockés à la fois dans le dépôt 2 et le dépôt 4

```
SELECT nom
FROM produits p, stocks s
WHERE p.prod_id = s.prod_id
AND depot_id = 2
AND p.prod_id IN(
SELECT prod_id
FROM stocks
WHERE depot_id = 4)
```

Res1(prod_id, nom)

prod_id	nom
1	A3
3	stylo

Res2(prod_id, nom)

prod_id	nom
1	A3
2	crayon

nom
A3

Res(nom)

Différence - 1

- Pareil que pour l'Intersection
- S'utilise normalement de la façon suivante en SQL :

```
SELECT <liste d'attributs projetés>
FROM   <liste de relations>
WHERE  <liste de critères de sélection…>

MINUS

SELECT …
FROM …
WHERE …
```

Différence - 2

- Ce n’est pas le cas en MySQL…
- … à la place, utilise l’opérateur NOT IN pour calculer une différence entre 2 relations

Exemple Différence

- Noms des produits qui ne sont pas stockés dans le dépôt 2

```
SELECT nom
FROM produits
WHERE prod_id NOT IN(
SELECT p.prod_id
FROM produits p, stocks s
WHERE p.prod_id = s.prod_id
AND depot_id=2)
```

Res1(prod_id, nom)

prod_id	nom
1	A3
2	crayon
3	stylo
4	A4

Res2(prod_id)

prod_id
1
3

nom
crayon
A4

Res(nom)

Fonctions d’agrégation - suite

- Dans le 1er cours, on a vu plusieurs fonctions dites fonctions d’agrégation (COUNT, SUM, AVG, MAX/ MIN)

Exemple SUM - 1

- Quantité de produits stockés au total :

Stock(<u>prod_id, depot_id</u>, qte)

prod_id	depot_id	qte
p1	1	0
p3	2	9
p1	3	15
p2	4	20
p3	5	0
p1	4	5
p2	5	2
p3	3	30
p1	2	10

```
SELECT SUM(qte)
FROM   Stocks
```

Res(SUM(qte))

SUM(qte)
91

Exemple SUM - 2

- Quantité de p1 stocké au total :

Stock(prod_id, depot_id, qte)

prod_id	depot_id	qte
p1	1	0
p3	2	9
p1	3	15
p2	4	20
p3	5	0
p1	4	5
p2	5	2
p3	3	30
p1	2	10

```
SELECT prod_id, SUM(qte)
FROM   Stocks
WHERE prod_id = 1
```

Res(prod_id, SUM(qte))

prod_id	SUM(qte)
1	30

Exemple SUM - 3

- Quantité stocké au total pour chaque produit :

Stock(prod_id, depot_id, qte)

prod_id	depot_id	qte
p1	1	0
p3	2	9
p1	3	15
p2	4	20
p3	5	0
p1	4	5
p2	5	2
p3	3	30
p1	2	10

```
SELECT prod_id, SUM(qte)
FROM   Stocks
```

→ **ERROR**

Partitionnement d'une relation - 1

- Pour résoudre ce problème, besoin de partitionner (regrouper) les tuples pour une même valeur
  - Dans l'exemple, faudrait regrouper à l'aide de *prod_id*
- Se fait à l'aide de l'opérateur GROUP BY

Partitionnement d'une relation - 2

- **Syntaxe**
  - GROUP BY <liste d'attributs>
- **Principe**
  - **Regroupement horizontal** d'une relation, **selon** les **valeurs de l'attribut ou du groupe d'attributs** spécifiés
  - **Relation logiquement fragmentée** en groupes de tuples, où tous les tuples d'un groupe ont la même valeur pour la liste d'attributs

Exemple Group By

- Somme des quantités en stock de chaque produit :

Stock(<u>prod_id, depot_id</u>, qte)

prod_id	depot_id	qte
1	1	0
3	2	9
1	3	15
2	4	20
3	5	0
1	4	5
2	5	2
3	3	30
1	2	10

```
SELECT prod_id,
       SUM(qte)
FROM   Stocks
GROUP BY prod_id
```

prod_id	depot_i	qte
1	1	0
	2	10
	3	15
	4	5
2	4	20
	5	2
3	2	9
	3	30
	5	10

prod_id	SUM(qte)
1	30
2	22
3	39

Res(prod_id, SUM(qte))

Where et Group By

- Possible de combiner l'utilisation de WHERE avec GROUP BY

- Si la requête comporte une clause WHERE, les tuples ne vérifiant pas la condition sont exclus **AVANT** d'opérer le groupe

Exemple Where + Group By

- Somme des quantités en stock de chaque produit dans tous les dépôts sauf le dépot 3 :

Stock(<u>prod_id, depot_id</u>, qte)

prod_id	depot_id	qte
1	1	0
3	2	9
1	3	15
2	4	20
3	5	0
1	4	5
2	5	2
3	3	30
1	2	10

```
SELECT prod_id,
       SUM(qte)
FROM   Stocks
WHERE depot_id != 3
```

prod_id	depot_i	qte
1	1	0
	2	10
	4	5
2	4	20
	5	2
3	2	9
	5	10

prod_id	SUM(qte)
1	15
2	22
3	9

Res(prod_id, SUM(qte))

Restriction sur groupements - 1

- WHERE nous permet d'appliquer un critère de restriction sur les tuples

- Mais s'applique à chaque tuple indépendamment

- Peut avoir besoin de filtrer à partir de données résultant de SUM, COUNT…

- Pour répondre à des questions comme "Quels sont les produits stockés dans plus de 2 dépôts ?"

Restriction sur groupements - 2

- **Syntaxe**
  - HAVING <condition de restriction>
- **Principe**
  - Ne conserve dans le résultat final que les groupements pour lesquels la condition de restriction est vérifiée

Exemple Having

- Produits stockés dans plus de 2 dépôts :

Stock(<u>prod_id, depot_id</u>, qte)

prod_id	depot_id	qte
1	1	0
3	2	9
1	3	15
2	4	20
3	5	0
1	4	5
2	5	2
3	3	30
1	2	10

```
SELECT prod_id
FROM   Stocks
GROUP BY prod_id
HAVING COUNT(*) > 2
```

prod_id	depot_i	qte
1	1	0
	2	10
	3	15
	4	5
2	4	20
	5	2
3	2	9
	3	30
	5	10

prod_id
1
3

Res(prod_id)

Ordonnancement des résultats - 1

- Peut vouloir trier les résultats de la requête avant de les fournir à l'utilisateur
- Par exemple pour afficher ses contacts par ordre alphabétique, ou les produits du moins cher au plus cher…

Ordonnancement des résultats - 2

- **Syntaxe**
  - ORDER BY <attribut1> [ASC/DESC [,<attribut2>…]]
- **Principe**
  - Trie les résultats
  - ASC permet de trier dans l'ordre croissant, optionnel
  - DESC dans l'ordre décroissant
  - Peut lister plusieurs attributs pour générer l'ordre

Exemple Order By

- Stocks triés par produits et quantité :

SELECT *

FROM stocks

ORDER BY prod_id ASC,

qte DESC

Stock(<u>prod_id, depot_id</u>, qte)

prod_id	depot_id	qte
1	1	0
3	2	9
1	3	15
2	4	20
3	5	0
1	4	5
2	5	2
3	3	30
1	2	10

Res(prod_id, depot_id, qte)

prod_id	depot_id	qte
1	3	15
1	2	10
1	4	5
1	1	0
2	4	20
2	5	2
3	3	30
3	2	9
3	5	0

Synthèse

- Syntaxe de base d'une requête SQL :

SELECT <liste d'attributs projetés>

FROM <liste de relations>

WHERE <liste de critères de sélection et de jointure>

Synthèse - 2

- Syntaxe avancée d'une requête SQL :

SELECT <liste attributs Aj et/ou expressions sur attributs Ap>

FROM <liste relations Ri>

WHERE <condition C1 sur tuples>

GROUP BY <liste attributs Ak>

HAVING < condition C2 sur groupes>

ORDER BY <liste attributs Al>

(6) SELECT <liste attributs Aj et/ou expressions sur attributs Ap>

(1) FROM <liste relations Ri>

(2) WHERE <condition C1 sur tuples>

(3) GROUP BY <liste attributs Ak>

(4) HAVING < condition C2 sur groupes>

(5) ORDER BY <liste attributs Al>

1. Produit cartésien des relations Ri
2. Sélection des tuples de (1) vérifiant C1
3. Partitionnement de l'ensemble obtenu en (2) en suivant les valeurs des Ak
4. Sélection des groupes de (3) vérifiant C2
5. Tri des groupes obtenus en (4) suivant les valeurs des Al
6. Projection de l'ensemble obtenu en (5) sur les attributs Aj, avec calcul des fonctions appliquées aux groupes

Prédicat d’existence - 1

- **Syntaxe**
  - EXISTS <sous-requête>
- **Principe**
  - Si l’ensemble résultat n’est pas vide, retourne vrai
  - Sinon retourne faux

Prédicat d'existence - 2

- Donner l'adresse des dépôts où est stocké le produit 2

Stock(<u>prod_id, depot_id</u>, qte)

prod_id	depot_id	qte
1	1	0
3	2	9
1	3	15
2	4	20
3	5	0
1	4	5
2	5	2
3	3	30
1	2	10

```
SELECT d.depot_id, adr
FROM depots d
WHERE EXISTS (SELECT *
    FROM stocks s
    WHERE s.prod_id = 2
     AND s.depot_id = d.depot_id)
```

Res(depot_id, adr)

depot_id	adr
4	Nancy
5	Nancy

Prédicat d’existence - 3

- Peut être nié : NOT EXISTS
- Adresses des dépôts où n’est pas stocké le produit 2

Stock(<u>prod_id, depot_id</u>, qte)

prod_id	depot_id	qte
1	1	0
3	2	9
1	3	15
2	4	20
3	5	0
1	4	5
2	5	2
3	3	30
1	2	10

```
SELECT d.depot_id, adr
FROM depots d
WHERE NOT EXISTS (SELECT *
    FROM stocks s
    WHERE s.prod_id = 2
     AND s.depot_id = d.depot_id)
```

Res(depot_id, adr)

depot_id	adr
1	Nancy
2	Laxou
3	Vandoeuvre

Division - 1

- Récupère les tuples de la première table qui sont en relation avec tous les tuples de la seconde table

- Liste des dépôts qui contiennent tous les produits

Stock(prod_id, depot_id, qte)

prod_id	depot_id	qte
1	1	0
3	2	9
1	3	15
2	4	20
3	5	0
1	4	5
2	5	2
3	3	30
1	2	10
1	5	7
4	5	42

Produits(prod_id, nom, pu)

prod_id	nom	pu
1	A3	10.0
2	crayon	9
3	stylo	15
4	A4	10.0

DIV(
PROJECT(Stock, prod_id, depot_id),
PROJECT(Produit, prod_id)
)

→

depot_id
5

Division - 2

- En SQL, **pas d'opérateur spécifique**
- À la place, **utilise** l'opérateur **EXISTS**
- Plus précisément, **utilise** à la place une **double négation** de **EXISTS**

Division - 3

- Liste des dépôts qui contiennent tous les produits
- Se paraphrase en français en :
    - “Quels sont les dépôts pour lesquels il **n’existe aucun produit** qui **n’est pas stocké** par ce dépôt ?”

Division - 4

- “Quels sont les dépôts pour lesquels il **n’existe aucun produit** qui **n’est pas stocké** par ce dépôt ?”

```
SELECT d.depot_id, adr
FROM   depots d
WHERE NOT EXISTS (
   SELECT *
   FROM produits p
   WHERE NOT EXISTS (
      SELECT *
      FROM stocks s
      WHERE p.prod_id = s.prod_id
      AND d.depot_id = s.depot_id ) )
```

depot_id
5

Division - 5

- En vrai, **se fait** très bien **avec** des les **opérateurs et fonctions d'agrégation** sinon
- **Intuition :**
    - "Suffit" de garder les dépôts pour qui stockent **autant de produits qu'il y en a au total** dans la table Produits

Des questions ?