Bases de données : requêtes imbriquées

Sélection

Une condition de sélection utilisée dans une clause `WHERE` s'exprime sous la forme d'une **comparaison** entre

Sélection

Une condition de sélection utilisée dans une clause `WHERE` s'exprime sous la forme d'une **comparaison** entre

- la **valeur d'un attribut** (ou une expression)

Sélection

Une condition de sélection utilisée dans une clause `WHERE` s'exprime sous la forme d'une **comparaison** entre

- la **valeur d'un attribut** (ou une expression)
- et **une valeur de référence**, qui, dans le cas des requêtes simples, est

Sélection

Une condition de sélection utilisée dans une clause `WHERE` s'exprime sous la forme d'une **comparaison** entre

- la **valeur d'un attribut** (ou une expression)
- et **une valeur de référence**, qui, dans le cas des requêtes simples, est
  - soit un attribut (ou une expression),

Sélection

Une condition de sélection utilisée dans une clause `WHERE` s'exprime sous la forme d'une **comparaison** entre

- la **valeur d'un attribut** (ou une expression)
- et **une valeur de référence**, qui, dans le cas des requêtes simples, est
  - soit un attribut (ou une expression),
  - soit une constante

Exemple

Dans la base de données `World`, le schéma de la table `City` est

Exemple

Dans la base de données `World`, le schéma de la table `City` est

Field	Type	Null	Key	Default	Extra
ID	int(11)	NO	PRI	NULL	AUTOINCREMENT
Name	char(35)	NO			
CountryCode	char(3)	NO			
District	char(20)	NO			
Population	int(11)	NO		0	

Exemple 1

- pour trouver le nom et le nombre d'habitants des villes de plus d'un million d'habitants :

Exemple 1

- pour trouver le nom et le nombre d'habitants des villes de plus d'un million d'habitants :

```
SELECT Name, Population
FROM City
WHERE Population >= 1000000;
```

Exemple 1

- pour trouver le nom et le nombre d'habitants des villes de plus d'un million d'habitants :

```
SELECT Name, Population
FROM City
WHERE Population >= 1000000;
```

- Et si on veut trouver le nom et le nombre d'habitants de la ville ayant le plus grand nombre d'habitants ?

Exemple 1

- pour trouver le nom et le nombre d'habitants des villes de plus d'un million d'habitants :

```
SELECT Name, Population
FROM City
WHERE Population >= 1000000;
```

- Et si on veut trouver le nom et le nombre d'habitants de la ville ayant le plus grand nombre d'habitants ?

```
SELECT Name, Population
FROM City
WHERE Population = MAX(Population);
```

Exemple 1

- pour trouver le nom et le nombre d'habitants des villes de plus d'un million d'habitants :

```
SELECT Name, Population
FROM City
WHERE Population >= 1000000;
```

- Et si on veut trouver le nom et le nombre d'habitants de la ville ayant le plus grand nombre d'habitants ?

```
SELECT Name, Population
FROM City
WHERE Population = MAX(Population);
```

```
Error Code: 1111. Invalid use of group function
```

Exemple 1

- pour trouver le nom et le nombre d'habitants des villes de plus d'un million d'habitants :

```
SELECT Name, Population
FROM City
WHERE Population >= 1000000;
```

- Et si on veut trouver le nom et le nombre d'habitants de la ville ayant le plus grand nombre d'habitants ?

```
SELECT Name, Population
FROM City
WHERE Population = MAX(Population);
```

`Error Code: 1111. Invalid use of group function`
$=>$ la fonction agrégative `MAX` ne peut être mise ici, seulement à coté du `SELECT`.
On trouve le maximum puis relance une requête en recherchant cette valeur : deux requêtes successives.

Exemple 1

Autre tentative alors en mettant la fonction agrégative à côté du SELECT

```
SELECT Name, MAX(Population) FROM City;
```

- ne fonctionne pas non plus...

Exemple 1

Autre tentative alors en mettant la fonction agrégative à côté du SELECT

```
SELECT Name, MAX(Population) FROM City;
```

- ne fonctionne pas non plus...
- car rend pour Name celle de la première ville de la table...

Exemple 1

Autre tentative alors en mettant la fonction agrégative à côté du SELECT

```
SELECT Name, MAX(Population) FROM City;
```

- ne fonctionne pas non plus...
- car rend pour Name celle de la première ville de la table...
- et pas le nom de la ville qui a la population maximale !

Exemple 2

- Comment trouver toutes les villes qui ont le même champ District qu'une ville donnée, en une seule requête ?

Exemple 2

- Comment trouver toutes les villes qui ont le même champ District qu'une ville donnée, en une seule requête ?
- Par exemple pour trouver les autres villes dans le même District (=Région) que celui de Paris ?

Exemple 2

- Comment trouver toutes les villes qui ont le même champ District qu'une ville donnée, en une seule requête ?
- Par exemple pour trouver les autres villes dans le même District (=Région) que celui de Paris ?
- En deux requêtes :

Exemple 2

- Comment trouver toutes les villes qui ont le même champ District qu'une ville donnée, en une seule requête ?
- Par exemple pour trouver les autres villes dans le même District (=Région) que celui de Paris ?
- En deux requêtes :
- `SELECT District FROM City WHERE Name = "Paris";`

Exemple 2

- Comment trouver toutes les villes qui ont le même champ District qu'une ville donnée, en une seule requête ?
- Par exemple pour trouver les autres villes dans le même District (=Région) que celui de Paris ?
- En deux requêtes :
- `SELECT District FROM City WHERE Name = "Paris";`
- rend "Île-de-France"

Exemple 2

- Comment trouver toutes les villes qui ont le même champ District qu'une ville donnée, en une seule requête ?
- Par exemple pour trouver les autres villes dans le même District (=Région) que celui de Paris ?
- En deux requêtes :
- `SELECT District FROM City WHERE Name = "Paris";`
- rend "Île-de-France"
- puis `SELECT Name FROM City WHERE District = "Ile-de-France";`

Exemple 2

- Comment trouver toutes les villes qui ont le même champ District qu'une ville donnée, en une seule requête ?
- Par exemple pour trouver les autres villes dans le même District (=Région) que celui de Paris ?
- En deux requêtes :
- `SELECT District FROM City WHERE Name = "Paris";`
- rend "Île-de-France"
- puis `SELECT Name FROM City WHERE District = "Ile-de-France";`
- rend Montreuil, Argenteuil, Paris et Boulogne-Billancourt

Exemple 2

- Solution possible (peu naturelle) : une auto-jointure

Exemple 2

- Solution possible (peu naturelle) : une auto-jointure
- ```
  SELECT City.Name
  FROM City
  JOIN City AS City2
  ON City.District = City2.District
  WHERE City2.Name = "Paris";
  ```
  ```

Exemple 3

- Comment trouver toutes les villes appartenant à un District ayant une ville de plus de 9 millions d'habitants ?

Exemple 3

- Comment trouver toutes les villes appartenant à un District ayant une ville de plus de 9 millions d'habitants ?
- En plusieurs requêtes :

Exemple 3

- Comment trouver toutes les villes appartenant à un District ayant une ville de plus de 9 millions d'habitants ?
- En plusieurs requêtes :
- Trouver les villes et les districts correspondant pour les villes de plus de 9 millions d'habitants

Exemple 3

- Comment trouver toutes les villes appartenant à un District ayant une ville de plus de 9 millions d'habitants ?
- En plusieurs requêtes :
- Trouver les villes et les districts correspondant pour les villes de plus de 9 millions d'habitants
- Chercher pour chaque district ainsi trouvé, toutes les villes de ce district...

Exemple 3

- Comment trouver toutes les villes appartenant à un District ayant une ville de plus de 9 millions d'habitants ?
- En plusieurs requêtes :
- Trouver les villes et les districts correspondant pour les villes de plus de 9 millions d'habitants
- Chercher pour chaque district ainsi trouvé, toutes les villes de ce district...
- Bref... on aimerait un moyen plus direct de répondre aux 3 exemples précédents.

Sous-requête

- Le langage SQL étend la notion de valeur de référence

Sous-requête

- Le langage SQL étend la notion de valeur de référence au résultat d'une requête.

Sous-requête

- Le langage SQL étend la notion de valeur de référence au résultat d'une requête.
- Ainsi, l'expression d'une clause `WHERE` peut prendre la forme :

Sous-requête

- Le langage SQL étend la notion de valeur de référence au résultat d'une requête.
- Ainsi, l'expression d'une clause `WHERE` peut prendre la forme : `... WHERE attribut opérateur (SELECT ...)`

Sous-requête

- Le langage SQL étend la notion de valeur de référence au résultat d'une requête.
- Ainsi, l'expression d'une clause `WHERE` peut prendre la forme : `... WHERE attribut opérateur (SELECT ...)`
- La requête, celle entre les parenthèses (qui sont **obligatoires**), dont le résultat sert de valeur de référence,

Sous-requête

- Le langage SQL étend la notion de valeur de référence au résultat d'une requête.
- Ainsi, l'expression d'une clause `WHERE` peut prendre la forme : `... WHERE attribut opérateur (SELECT ...)`
- La requête, celle entre les parenthèses (qui sont **obligatoires**), dont le résultat sert de valeur de référence, est une **requête imbriquée** ou une **sous-requête**

Sous-requête

- Le langage SQL étend la notion de valeur de référence au résultat d'une requête.
- Ainsi, l'expression d'une clause `WHERE` peut prendre la forme : `... WHERE attribut opérateur (SELECT ...)`
- La requête, celle entre les parenthèses (qui sont **obligatoires**), dont le résultat sert de valeur de référence, est une **requête imbriquée** ou une **sous-requête**
  La requête qui utilise ce résultat est la **requête principale**

Sous-requête

- Le langage SQL étend la notion de valeur de référence au résultat d'une requête.
- Ainsi, l'expression d'une clause `WHERE` peut prendre la forme : `... WHERE attribut opérateur (SELECT ...)`
- La requête, celle entre les parenthèses (qui sont **obligatoires**), dont le résultat sert de valeur de référence, est une **requête imbriquée** ou une **sous-requête**
  La requête qui utilise ce résultat est la **requête principale**
- On peut imbriquer autant de requêtes que l'on veut (attention à la lisibilité...)

Les opérateurs possibles

Les opérateurs possibles diffèrent suivant que la sous-requête renvoie

Les opérateurs possibles

Les opérateurs possibles diffèrent suivant que la sous-requête renvoie

- exactement une valeur

Les opérateurs possibles

Les opérateurs possibles diffèrent suivant que la sous-requête renvoie

- exactement une valeur
- ou toute une colonne

Sous-requête renvoyant une seule valeur

- on peut alors utiliser comme opérateur :

Sous-requête renvoyant une seule valeur

- on peut alors utiliser comme opérateur :
  $=$, $<$, $>$, $<=$, $>=$, $!=$, etc.

Sous-requête renvoyant une seule valeur

- on peut alors utiliser comme opérateur :
  $=$, $<$, $>$, $<=$, $>=$, $!=$, etc.
- et un des termes de la comparaison est le résultat de la sous requête.

Sous-requête renvoyant une seule valeur

- on peut alors utiliser comme opérateur :
  $=$, $<$, $>$, $<=$, $>=$, $!=$, etc.
- et un des termes de la comparaison est le résultat de la sous requête.
- On peut répondre alors à l'exemple 1 :

Sous-requête renvoyant une seule valeur

- on peut alors utiliser comme opérateur :
  $=$, $<$, $>$, $<=$, $>=$, $!=$, etc.
- et un des termes de la comparaison est le résultat de la sous requête.
- On peut répondre alors à l'exemple 1 :

```
SELECT Name, Population FROM City
WHERE Population = (SELECT MAX(Population)FROM
    City);
```

Sous-requête renvoyant une seule valeur

- on peut alors utiliser comme opérateur :
  $=$, $<$, $>$, $<=$, $>=$, $!=$, etc.
- et un des termes de la comparaison est le résultat de la sous requête.
- On peut répondre alors à l'exemple 1 :

```
SELECT Name, Population FROM City
WHERE Population = (SELECT MAX(Population)FROM
    City);
```

- pour l'exemple 2

Sous-requête renvoyant une seule valeur

- on peut alors utiliser comme opérateur :
  $=, <, >, <=, >=, !=$, etc.
- et un des termes de la comparaison est le résultat de la sous requête.
- On peut répondre alors à l'exemple 1 :

```
SELECT Name, Population FROM City
WHERE Population = (SELECT MAX(Population)FROM
    City);
```

- pour l'exemple 2

```
SELECT Name FROM City
WHERE District = (SELECT District FROM City WHERE
     Name = "Paris");
```

Sous-requête renvoyant toute une colonne

On peut alors utiliser comme opérateur :

Sous-requête renvoyant toute une colonne

On peut alors utiliser comme opérateur :

- `IN`, `EXISTS` principalement (il y en a d'autres)

Sous-requête renvoyant toute une colonne

On peut alors utiliser comme opérateur :

- `IN`, `EXISTS` principalement (il y en a d'autres)
- qui permet de tester si une valeur est dans (`IN`) le résultat renvoyé par la sous requête

Sous-requête renvoyant toute une colonne

On peut alors utiliser comme opérateur :

- `IN`, `EXISTS` principalement (il y en a d'autres)
- qui permet de tester si une valeur est dans (`IN`) le résultat renvoyé par la sous requête
- ou de savoir si la sous requête a renvoyé au moins un résultat (`EXISTS`) (Dans ce cas la sous-requête peut même renvoyer plus qu'une colonne).

Sous-requête renvoyant toute une colonne

On peut alors utiliser comme opérateur :

- `IN`, `EXISTS` principalement (il y en a d'autres)
- qui permet de tester si une valeur est dans (`IN`) le résultat renvoyé par la sous requête
- ou de savoir si la sous requête a renvoyé au moins un résultat (`EXISTS`) (Dans ce cas la sous-requête peut même renvoyer plus qu'une colonne).
- qu'on peut combiner éventuellement avec `NOT`

Sous-requête renvoyant toute une colonne

On peut alors utiliser comme opérateur :

- `IN`, `EXISTS` principalement (il y en a d'autres)
- qui permet de tester si une valeur est dans (`IN`) le résultat renvoyé par la sous requête
- ou de savoir si la sous requête a renvoyé au moins un résultat (`EXISTS`) (Dans ce cas la sous-requête peut même renvoyer plus qu'une colonne).
- qu'on peut combiner éventuellement avec `NOT`
- Pour l'exemple 3 on peut ainsi proposer :

Sous-requête renvoyant toute une colonne

On peut alors utiliser comme opérateur :

- `IN`, `EXISTS` principalement (il y en a d'autres)
- qui permet de tester si une valeur est dans (`IN`) le résultat renvoyé par la sous requête
- ou de savoir si la sous requête a renvoyé au moins un résultat (`EXISTS`) (Dans ce cas la sous-requête peut même renvoyer plus qu'une colonne).
- qu'on peut combiner éventuellement avec `NOT`
- Pour l'exemple 3 on peut ainsi proposer :

```
SELECT Name FROM City
WHERE District IN (SELECT District FROM City
                   WHERE Population > 9000000)
                      ;
```

Sous-requête renvoyant toute une colonne

- On a déjà vu (sans trop comprendre alors...) ce mécanisme pour implémenter le `INTERSECT`

Sous-requête renvoyant toute une colonne

- On a déjà vu (sans trop comprendre alors...) ce mécanisme pour implémenter le `INTERSECT`
- Dans le cas où les deux relations ont deux attributs en commun :

```
SELECT a,b FROM R
WHERE EXISTS (SELECT c,d FROM S WHERE c=a AND d=b)
```

Sous-requête renvoyant toute une colonne

- On a déjà vu (sans trop comprendre alors...) ce mécanisme pour implémenter le `INTERSECT`
- Dans le cas où les deux relations ont deux attributs en commun :

```
SELECT a,b FROM R
WHERE EXISTS (SELECT c,d FROM S WHERE c=a AND d=b)
```

- On peut noter que les champs `a` et `b` de `R` sont connus dans la sous-requête ;

Table intermédiaire

- Il y a des situations dans lesquelles on aimerait stocker le résultat d'une requête dans une table intermédiaire pour pouvoir la réutiliser

Table intermédiaire

- Il y a des situations dans lesquelles on aimerait stocker le résultat d'une requête dans une table intermédiaire pour pouvoir la réutiliser
- Cela semble naturel car le résultat d'un `SELECT` **EST** une table ! (mais volatile...)

Table intermédiaire

- Il y a des situations dans lesquelles on aimerait stocker le résultat d'une requête dans une table intermédiaire pour pouvoir la réutiliser
- Cela semble naturel car le résultat d'un `SELECT` **EST** une table ! (mais volatile...)
- Les SGBD offrent cette possibilité

Table intermédiaire

- Il y a des situations dans lesquelles on aimerait stocker le résultat d'une requête dans une table intermédiaire pour pouvoir la réutiliser
- Cela semble naturel car le résultat d'un `SELECT` **EST** une table ! (mais volatile...)
- Les SGBD offrent cette possibilité
  Par exemple en MySQL : on peut remplacer ce qui est à droite d'une clause `FROM`

Table intermédiaire

- Il y a des situations dans lesquelles on aimerait stocker le résultat d'une requête dans une table intermédiaire pour pouvoir la réutiliser
- Cela semble naturel car le résultat d'un `SELECT` **EST** une table ! (mais volatile...)
- Les SGBD offrent cette possibilité
  Par exemple en MySQL : on peut remplacer ce qui est à droite d'une clause `FROM`
  - par une requête `(SELECT ... )`

Table intermédiaire

- Il y a des situations dans lesquelles on aimerait stocker le résultat d'une requête dans une table intermédiaire pour pouvoir la réutiliser
- Cela semble naturel car le résultat d'un `SELECT` **EST** une table ! (mais volatile...)
- Les SGBD offrent cette possibilité
  Par exemple en MySQL : on peut remplacer ce qui est à droite d'une clause `FROM`
  - par une requête (`SELECT ... `)
  - à condition de nommer la sous-requête, i.e de simplement rajouter un nom à la fin :

Table intermédiaire

- Il y a des situations dans lesquelles on aimerait stocker le résultat d'une requête dans une table intermédiaire pour pouvoir la réutiliser
- Cela semble naturel car le résultat d'un `SELECT` **EST** une table ! (mais volatile...)
- Les SGBD offrent cette possibilité
  Par exemple en MySQL : on peut remplacer ce qui est à droite d'une clause `FROM`
  - par une requête `(SELECT ... )`
  - à condition de nommer la sous-requête, i.e de simplement rajouter un nom à la fin :
  - `SELECT... FROM (SELECT ...)<NomTableIntermediaire>`

Table intermédiaire

- Il y a des situations dans lesquelles on aimerait stocker le résultat d'une requête dans une table intermédiaire pour pouvoir la réutiliser
- Cela semble naturel car le résultat d'un `SELECT` **EST** une table ! (mais volatile...)
- Les SGBD offrent cette possibilité
  Par exemple en MySQL : on peut remplacer ce qui est à droite d'une clause `FROM`
  - par une requête `(SELECT ... )`
  - à condition de nommer la sous-requête, i.e de simplement rajouter un nom à la fin :
  - `SELECT... FROM (SELECT ...)<NomTableIntermediaire>`

  On a alors des requêtes lourdes et peu lisibles...mais on ne peut pas faire autrement...