LIF4 - Initiation aux Bases de données : SQL - 1

E.Coquery

emmanuel.coquery@liris.cnrs.fr

http ://liris.cnrs.fr/∼ecoquery

SQL

- Un langage concret interagir avec le modèle relationnel :
  - Un langage de manipulation de données.
  - Un langage de description de données.
  - Un langage pour administrer la base, gérer les contrôles d'accès.
- Origine : IBM, dans les années 70.
- Standards :
  - SQL-87 : 1987 (ISO)
  - SQL-2 : 1992 (ANSI)
  - SQL-3 : 1999
  - SQL-2003
  - SQL-2006
- Différences avec la théorie :
  - possibilités de doublons ;
  - possibilité d'ordonner le résultat des requêtes ;
  - notion de valeur non définie.

Interrogation simple

SELECT att_1, att_2, ...
FROM *nom_table* ;

- Récupérer les valeurs contenus dans la table `nom_table`, en ne gardant que les attributs `att1, att2, ...`
- En algèbre relationnelle :
  $\pi_{att_1,att_2,\ldots}(nom\_table)$
- En calcul relationnel "tuple" :
  $\{t.att_1, t.att_2, \cdots \mid nom\_table(t)\}$

On peut remplacer att_1, att_2, ... par $*$ pour utiliser tous les attributs.

Exemple

Schéma :

Employe(Nom, Num, Fonction, Num_sup, Embauche, Salaire, Num_Dept)

Donner le nom et la fonction de chaque employé :

- `SELECT Nom,Fonction FROM Employe ;`
- $\pi_{Nom,Fonction}(Employe)$
- $\{t.Nom, t.Fonction \mid Employe(t)\}$

Demo

Exemple 2

Schéma :

Employe(Nom, Num, Fonction, Num_sup, Embauche, Salaire, Num_Dept)

Donner les informations sur chaque employé :

- `SELECT * FROM Employe ;`
- *Employe*
- $\{t.Nom, t.Num, t.Fonction, t.Num\_sup, t.Embauche, t.Salaire, t.Num\_Dept \mid Employe(t)\}$

Demo

mot clé DISTINCT

Le mot clé DISTINCT permet d'éliminer les doublons dans le résultat.

Exemple :

Donner les différentes fonctions occupées dans l'entreprise :

- `SELECT DISTINCT Fonction FROM Employe ;`

Demo

Sélections (de lignes)

SELECT att_1, att_2, ...
FROM *nom_table*
WHERE *condition*

- La clause WHERE spécifie les lignes à sélectionner grâce à la *condition*.
- En algèbre relationnelle :
  $\pi_{att_1,att_2,...}(\sigma_{condition}(nom\_table))$
- En calcul relationnel "tuple" :
  $\{t.att_1, t.att_2, \cdots \mid nom\_table(t) \wedge condition\}$

Conditions du WHERE

Expressions simples :

- Comparaisons (=, !=, <, <=, >, >=)
- entre un attribut et une constante ou un autre attribut
- différents types de données utilisés pour les constantes :
  - nombres : 1, 1980, 1.5
  - chaînes de caractères : 'Martin', 'directeur'
  - dates : '1980-06-18'
    - le formattage des dates peut varier

Combinaison d'expressions via :

- le $\wedge$: AND
- le $\vee$: OR

Exemple

Schéma :

Employe(Nom, Num, Fonction, Num_sup, Embauche, Salaire, Num_Dept)

Quels sont les employés dont la date d'embauche est antérieure au 1^{er} janvier 1999 :

- ```
  SELECT Nom
  FROM Employe
  WHERE Embauche < '1999-01-01' ;
  ```
- $\pi_{Nom}(\sigma_{Embauche<'1999-01-01'}(Employe))$
- $\{t.Nom \mid Employe(t) \wedge t.Embauche <' 1999-01-01'\}$

Demo

Exemple 2

Schéma :

Employe(Nom, Num, Fonction, Num_sup, Embauche, Salaire, Num_Dept)

Quels sont les employés dont la date d'embauche est antérieure au 1^{er} janvier 1999 et touchant au moins 30000 euros de salaire :

- ```
  SELECT Nom
  FROM Employe
  WHERE Embauche < '1999-01-01'
  AND Salaire >= 30000 ;
  ```
- $\pi_{Nom}(\sigma_{Embauche<'1999-01-01'}(Employe))$
- $\{t.Nom \mid Employe(t) \wedge t.Embauche <' 1999-01-01'\}$

Demo

Autres conditions

- L'opérateur IN permet de spécifier un ensemble de valeur possibles :
  - Quels sont les employés qui sont directeur ou ingénieur ?
    ```
    SELECT Nom, Fonction
    FROM Employe
    WHERE Fonction IN ('ingenieur','directeur') ;
    ```
- L'opérateur BETWEEN ... AND permet de spécifier un intervalle de valeurs :
  - Quels employés gagnent entre 25000 et 30000 euros ?
    ```
    SELECT Nom, Salaire
    FROM Employe
    WHERE Salaire BETWEEN 25000 AND 30000 ;
    ```
  - Attention à ne pas confondre le AND du BETWEEN avec celui qui correspond au $\wedge$.

Autre exemple

Quels sont les employés directeur ou ingénieur, embauchés entre le 1^{er} janvier 1990 et le 31 décembre 1999 gagnant moins de 32000 euros ?

```
SELECT Nom, Embauche, Fonction, Salaire
FROM Employe
WHERE Fonction IN ('ingenieur','directeur')
AND Embauche BETWEEN '1990-01-01' AND '1999-12-31'
AND Salaire < 32000 ;
```

condition, connecteur $\wedge$

Valeurs non définies

En pratique, il est possible d'avoir des valeurs non définies.

- Elles sont représentées par le mot clé NULL.
- On peut tester si une valeur n'est pas définie grâce à la condition IS NULL (ou au contraire IS NOT NULL)

Schéma : Batiment(Num_bat, Nom_bat, Ent_princ, Ent_Sec)

- Les bâtiments qui n'ont pas d'entrée secondaire auront une valeur NULL pour l'attribut Ent_Sec.
- La requête suivante indique les bâtiments n'ayant pas d'entrée secondaire :

```
SELECT *
FROM Batiment
WHERE Ent_sec IS NULL ;
```


Tri du résultat d'une requête

En pratique, il peut être intéressant de trier le résultat d'un requête.

SELECT att_1, att_2, ...
FROM *nom_table*
WHERE *condition*
ORDER BY att_i, att_j, ...

- Le résultat de la requête est trié par ordre croissant sur l'attribut att_i
- En cas d'égalité entre deux lignes au niveau de l'attribut att_i, on utilise l'attribut att_j, etc ...
- Dans un ORDER BY, il est possible de faire suivre le nom d'un attribut par ASC ou DESC pour indiquer un ordre croissant ou décroissant.

Exemple

Schéma :

Employe(Nom, Num, Fonction, Num_sup, Embauche, Salaire, Num_Dept)

Donner le nom des employés du département numéro 20, en triant le résultat par salaire décroissant, puis par nom (croissant) :

```
SELECT Nom
FROM Employe
WHERE Num_dept=20
ORDER BY Salaire DESC, Nom ;
```


Requêtes sur plusieurs tables

SELECT att_1, att_2, ...
FROM nom_table_1, nom_table_2, ...
WHERE *condition*
ORDER BY att_i, att_j, ...

- Il est possible d'utiliser plusieurs tables dans une requête.
- Cela correspond à effectuer un produit cartésien entre les différentes tables.
- Si un attribut est présent dans plusieurs tables utilisées, on doit l'écrire *nom_table.att*

Jointures

En SQL, la jointure s'exprime comme une sélection sur le produit cartésien.

Jointure naturelle sur les relations $R(A_1, A_2, B_1, B_2)$ et $S(C_1, C_2, B_1, B_2)$ peut s'exprimer par :

SELECT A_1, A_2, $R.B_1$, $S.B_2$, C_1, C_2
FROM R, S
WHERE $R.B_1$=$S.B_1$ AND $R.B_2$=$S.B_2$

Exemple

Schéma :
Batiment(Num_bat, Nom_bat, Ent_princ, Ent_Sec)
Departement(Num_dept, Nom_dept, Num_bat)

Donner les départements avec leur bâtiments :

- *Departement* $\bowtie$ *Batiment*
-
```
SELECT Num_dept, Nom_dept, Batiment.Num_bat,
    Nom_bat, Ent_princ, Ent_sec
FROM Departement, Batiment
WHERE Departement.Num_bat = Batiment.Num_bat ;
```

Une notation similaire peut être utilisée pour renommer les attributs dans le SELECT.

Renommages

Il est parfois utile de renommer des tables :

SELECT att_1, att_2, ...
FROM nom_table_1 $nouveau\_nom_1$,
nom_table_2 $nouveau\_nom_2$, ...
WHERE *condition*
ORDER BY att_i, att_j, ...

- Indication des renommage dans le FROM.
- Les anciens noms indiqués dans le FROM ne peuvent pas être utilisés dans les autres parties de la requête.
- Utile lorsque l'on veut effectuer des jointures ou des produits cartésiens d'une table avec elle-même.

Exemple

Schema :
Employe(Nom, Num, Fonction, Num_sup, Embauche, Salaire, Num_Dept)

Donner les noms et la fonction des employés avec le nom de leur supérieur hiérarchique.

- $\pi_{Nom,Superieur,Fonction}(\sigma_{Num=Num\_sup}(\pi_{Nom,Num\_sup,Fonction}(Employe) \times \pi_{Superieur,Num}(\rho_{Nom/Superieur}(Employe))))$
- ```
  SELECT Employe.Nom, Employe.Fonction,
         Chef.Nom Superieur
  FROM Employe, Employe Chef
  WHERE Chef.Num = Employe.Num_sup ;
  ```


Exemple 2

Schema :
Employe(Nom, Num, Fonction, Num_sup, Embauche, Salaire, Num_Dept)

Quels sont les employés, donnés avec leur salaire, qui gagnent moins que Bellot ?

```
SELECT Employe.Nom, Employe.Salaire
FROM Employe, Employe bel
WHERE Employe.Salaire < bel.Salaire
AND bel.Nom = 'Bellot' ;
```


Sous-requêtes

Il est possible d'utiliser le résultat d'une requête dans une autre requête.

- Augmentation de la puissance d'expression du langage.
- Les sous-requêtes sont utilisables dans les parties
  - WHERE
  - FROM (à condition de renommer le résultat)
  - SELECT (à condition que pour chaque ligne sélectionnée par la requête principale, on ne selectionne qu'une ligne dans la sous-requête).
- En cas de conflit sur les nom, c'est la déclaration la plus proche qui est utilisée.

Exemple

Si la sous-requête renvoie un résultat simple sur une ligne :

Schéma :
Employe(Nom, Num, Fonction, Num_sup, Embauche, Salaire, Num_Dept)

Quels sont les employés ayant la même fonction que 'Jones' ?

```
SELECT Nom
FROM Employe
WHERE Fonction =
       (SELECT Fonction
        FROM Employe
        WHERE Nom='Jones') ;
```


Exemple : Sous-requête liée à la requête principale

Schéma :
Employe(Nom, Num, Fonction, Num_sup, Embauche, Salaire, Num_Dept)

Quels sont les employés qui ne travaillent pas dans le même département que leur supérieur ?

```
SELECT Nom
FROM Employe Emp
WHERE Num_dept !=
       (SELECT Num_dept
        FROM Employe
        WHERE Emp.Num_sup = Num) ;
```
```

Sous-requêtes renvoyant plusieurs lignes

Opérateurs permettant d'utiliser de telles sous-requêtes :

- *a* IN (*sous_requete*)
  - vrai si *a* apparaît dans le résultat de *sous_requete*.
- *a* □ ANY (*sous_requete*)
  où □ peut être $\{=, <, >, <=, >=\}$
  - vrai si il existe un *b* parmi les lignes renvoyées par *sous_requete* tel que *a*□*b* soit vrai.
- *a* □ ALL (*sous_requete*)
  où □ peut être $\{=, <, >, <=, >=\}$
  - vrai si pour toutes les lignes *b* renvoyées par *sous_requete*, *a*□*b* est vrai.
- EXISTS (*sous_requete*)
  - vrai si le résultat de *sous_requete* n'est pas vide.

Exemple

Schéma :
Employe(Nom, Num, Fonction, Num_sup, Embauche, Salaire, Num_Dept)

Quels sont les employés, donnés avec leur salaire, gagnant plus que tous les employés du département 20 ?

```
SELECT Nom, Salaire
FROM Employe
WHERE Salaire > ALL (SELECT Salaire
                     FROM Employe
                     WHERE Num_dept = 20) ;
```


Exemple 2

Schéma :
Employe(Nom, Num, Fonction, Num_sup, Embauche, Salaire, Num_Dept)

Quels sont les employés qui ont un subalterne ?

```
SELECT Nom
FROM Employe Chef
WHERE EXISTS (SELECT Nom
              FROM Employe
              WHERE Employe.Num_sup = Chef.Num) ;
```


Sous-requête avec un résultat à plusieurs colonnes

On peut utiliser la notation (*a*, *b*, ...) pour former un n-uplet à comparer avec le résultat de la sous-requête :

Schéma :
Employe(Nom, Num, Fonction, Num_sup, Embauche, Salaire, Num_Dept)

Quels sont les employés ayant même fonction et même supérieur que 'Bellot' ?

```
SELECT Nom
FROM Employe
WHERE (Fonction, Num_sup) = (SELECT Fonction, Num_sup
                             FROM Employe
                             WHERE Nom='Bellot') ;
```


Sous-requêtes imbriquées

Il est possible d'imbriquer les sous-requêtes :

Employe(Nom, Num, Fonction, Num_sup, Embauche, Salaire, Num_Dept)

Donner le nom et la fonction des employés du département 20 ayant même fonction qu'une personne du département de 'Dupont'.

```
SELECT Nom, Fonction
FROM Employe
WHERE Num_dept = 20
AND fonction IN
        (SELECT Fonction
         FROM Employe
         WHERE Num_dept = (SELECT Num_dept
                           FROM Employe
                           WHERE Nom = 'Dupont')) ;
```