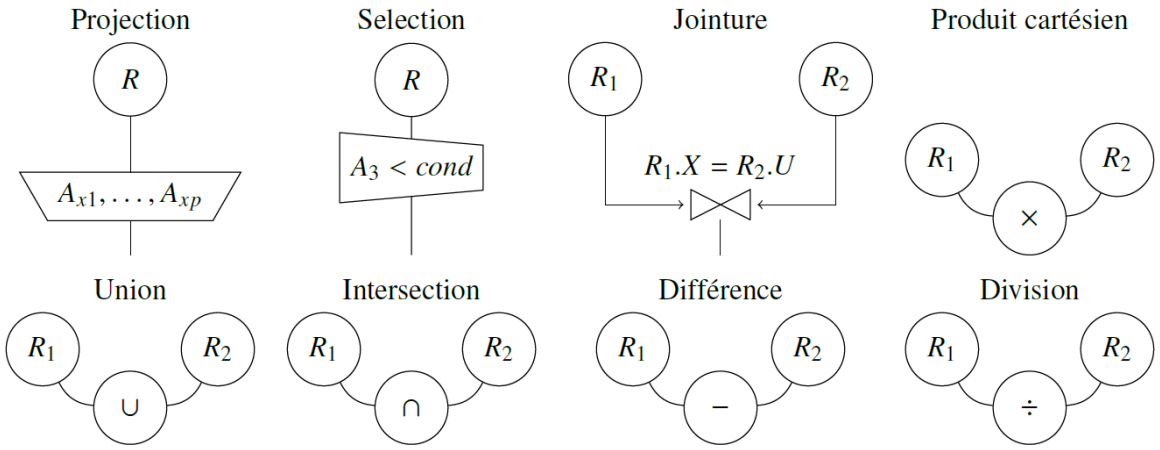
BASES DE DONNÉES : COMPLÉMENTS



Objectifs

A la fin de la séquence d'enseignement l'élève doit pouvoir :

- utiliser une application offrant une interface graphique pour créer une base de données et l'alimenter
- utiliser une application offrant une interface graphique pour lancer des requêtes sur une base de données
- distinguer les rôles respectifs des machines client, serveur, et éventuellement serveur de données
- traduire dans le langage de l'algèbre relationnelle des requêtes écrites en langage courant
- concevoir une base constituée de plusieurs tables, et utiliser les jointures symétriques pour effectuer des requêtes croisées

Table des matières

1 Compléments de SQL

Ne faisant pas partie de l'algèbre relationnel, des commandes viennent compléter les requêtes SQL, par exemple pour trier les résultats ou effectuer des opérations des groupes de tuples.

1.1 Relation d'ordre

La clause `ORDER BY` permet de trier les résultats. Par défaut l'ordre est ascendant `ASC` mais avec `DESC` on peut spécifier l'ordre descendant du tri.

Quand plusieurs attributs sont sélectionnés, le tri des colonnes se fait d'abord selon la colonne du premier attribut, puis selon les suivantes en cas d'égalité des premières.

```
SELECT * FROM R1 ORDER BY att1 DESC, att2 ASC;
```

EXEMPLE : faire apparaître les départements triés ordre décroissant dans chaque département les numéros de station triés par ordre croissant :

```
SELECT departement,num_sta FROM radome ORDER BY departement DESC, num_sta ASC;
```

La clause `ORDER BY` se positionne en fin de requête SQL puisqu'elle permet de trier le résultat final et doit figurer d'une seule fois après la clause `SELECT`.

1.2 Fonction de sous-requêtes

1.2.1 Sous-requête renvoyant un unique élément

Grâce aux fonctions d'agrégation, nous avons vu (cours précédent) qu'il était possible d'effectuer des opérations qui, sur l'ensemble des valeurs d'un attribut d'une relation, renvoient un résultat composé d'un unique élément. Il est alors possible d'utiliser ce résultat dans une sélection :

```
SELECT ... FROM ... WHERE machin = (SELECT ... FROM ....) ;
```

EXEMPLE : déterminer le nom et l'altitude de la plus haute station météo de la relation `RADOME` :

```
SELECT nom_usuel,altitude FROM radome WHERE altitude=(SELECT max(altitude) FROM radome);
```

1.2.2 Sous-requête renvoyant plusieurs éléments

Si la sous-requêtes renvoie plusieurs éléments, on peut alors appliquer une sélection avec :

- `IN ...` c'est appartenance $\in$
- `NOT IN ...` c'est la non appartenance $\notin$
- `op ANY` c'est l'existence $\exists$ associée à un opérateur `op` où `op` est un opérateur comme par exemple =, !=, <...
- `op ALL` c'est le quelque soit $\forall$ associé à un opérateur `op` où `op` est un opérateur comme par exemple =, !=, <...

REMARQUES:

- `IN` produit le même effet que `= ANY`

- `NOT IN` produit le même effet que `!= ALL`

EXEMPLE : quelles sont les noms de villes dont la température était supérieure à 297 K le jour de la mesure ?

```
SELECT nom_usuel FROM radome WHERE id_omm IN (SELECT id_omm FROM synop WHERE t>295);
```

qui donne le même résultat que :

```
SELECT nom_usuel FROM radome ra JOIN synop sy  ON ra.id_omm=sy.id_omm WHERE t>295;
```

EXEMPLE : quelles sont les villes non situées en Bourgogne ?

```
SELECT nom_usuel  FROM radome WHERE departement NOT IN (21,58,71,89);
```

```
SELECT nom_usuel  FROM radome
WHERE departement NOT IN (SELECT departement FROM radome WHERE region="Bourgogne");
```

1.2.3 Cas de l'existence

La requête `EXISTS sous-req` donne un résultat vrai si la sous requête `sous-req` retourne au moins une ligne.

```
SELECT ... FROM ... WHERE EXISTS (SELECT ...);
```

Par ailleurs, la requête `NOT EXISTS sous-req` donne un résultat vrai si la sous requête `sous-req` ne retourne pas de ligne.

```
SELECT ... FROM ... WHERE NOT EXISTS (SELECT ...);
```

EXEMPLE : quels sont les noms des élèves n'étant pas seuls issus d'un lycée ?

```
SELECT nom FROM personne p JOIN eleve e ON e.num_p = p.num_p WHERE EXISTS
(SELECT * FROM eleve WHERE lycée_origine = e.lycée_origine AND num_p != e.num_p);
```

ou bien

```
SELECT nom FROM personne p JOIN eleve e ON e.num_p = p.num_p
WHERE 1 < (SELECT COUNT(*) FROM eleveWHERE lycée_origine = e.lycée_origine);
```

ou encore avec deux jointures :

```
SELECT nom FROM personne p JOIN eleve e1 ON e1.num_p = p.num_p
JOIN eleve e2 ON e2.lycée_origine = e1.lycée_origine WHERE e1.num_p != e2.num_p;
```

1.2.4 Cas de la division

Nous avons vu dans le cours précédent que le division ne fait pas actuellement partie de la norme SQL. Cependant, comment répondre à la question :
PROBLÉMATIQUE : Quels sont les numéros des professeurs enseignant à tous les étudiants ?
On peut reformuler le problème sous la forme :
PROBLÉMATIQUE : Quels sont les numéros des professeurs tels que quelque soit l'étudiant, cet étudiant est dans une des classes du professeur ?
Hélas, nouveau problème, si `EXISTS` ($\exists$) existe , `FORALL` ($\forall$) n'existe pas ! On s'en sort alors avec deux `NOT EXIST` imbriqués. En effet : $t \in R_1 \div R_2 \Leftrightarrow \forall t' \in R_2, (t,t') \in R_1 \Leftrightarrow \nexists t' \in R_2 \mid \nexists t'' \in R_1 \ ; \ t'' = (t,t')$ ce qui revient à dire :

PROBLÉMATIQUE :

- Quels sont les numéros des professeurs tels qu'il n'existe pas d'étudiants tel qu'il n'existe pas de cours de ce professeur suivi par cet étudiant ?

```
SELECT num_p FROM prof WHERE NOT EXISTS
        (SELECT * FROM eleve e WHERE NOT EXISTS
                (SELECT * FROM enseigne
                WHERE num_prof = num_p AND num_classe = e.num_classe
                )
        );
```

- Quels sont les numéros des professeurs tels qu'il n'existe pas d'étudiants qui ne soit pas dans l'une des classes de ce professeur ?

```
SELECT num_p FROM prof WHERE NOT EXISTS
        (SELECT * FROM eleve e WHERE num_classe NOT IN
                (SELECT num_classe FROM enseigne WHERE num_prof = num_p
                )
        );
```

ce qui pourra se faire de façon simple avec :

PROBLÉMATIQUE : Quels sont les numéros des professeurs tels que le nombre d'étudiant qui suivent un cours de ce professeur est le nombre total d'étudiants.

```
SELECT num_p FROM prof WHERE
        (SELECT COUNT(*) FROM eleve e)
        = (SELECT COUNT (*) FROM eleve e JOIN enseigne en ON e.num_classe = en.num_classe
                WHERE en.num_prof = num_p );
```

1.3 Regroupements avant agrégation

1.3.1 Regroupement par attributs communs

OBJECTIF : La clause `GROUP BY` permet de regrouper les tuples ayant certains attributs en commun en vue d'utiliser une fonction d'agrégat. Elle ne renvoie qu'une seule ligne par groupe.

```
SELECT ... FROM ... WHERE ... GROUP BY ... ;
```

ATTENTION ! il faut que le contenu du `SELECT` soit cohérent, i.e. :

- soit composé des attributs du `GROUP BY` ou qui sont identiques pour chaque élément du groupe
- soit des fonctions d'agrégat (qui porteront sur chaque groupe)

EXEMPLE : Quels sont les nombres de cours dispensés par chaque enseignant ?

```
SELECT nom, prénom, COUNT (*)
FROM enseigne en JOIN personne pe ON en.num_prof=pe.num_p
GROUP BY nom, prénom;
```

EXEMPLE : Quel est le nombre moyen d'élèves pour chaque professeur ?

```
SELECT nom,"Effectif moyen" FROM personne pe JOIN
        (SELECT num_prof, AVG(effectif) AS "Effectif moyen"
                FROM classe JOIN enseigne ON num_classe = no
                GROUP BY num_prof) gr
        ON pe.num_p=gr.num_prof;
```

ATTENTION ! si des professeurs n'ont pas d'élèves, ils n'apparaîtront pas. On peut les ajouter avec une `UNION`.

```
SELECT nom,"Effectif moyen" FROM personne pe JOIN
        (SELECT num_p, 0 AS "Effectif moyen" FROM prof
                WHERE num_p NOT IN (SELECT num_prof from enseigne)
        UNION
        SELECT num_prof, AVG(effectif) AS "Effectif moyen"
                FROM classe JOIN enseigne ON num_classe = no
                GROUP BY num_prof) gr
        ON pe.num_p=gr.num_p;
```

1.3.2 Conditions sur les regroupements par attributs communs

OBJECTIF : La clause `HAVING` permet d'imposer une condition sur les groupes définis par `GROUP BY`. C'est l'équivalent de la clause `WHERE` après la clause `SELECT`. C'est pour cela qu'elle se place après `GROUP BY` :

```
SELECT ... FROM ... WHERE ... GROUP BY ... HAVING ... ;
```

REMARQUE: Le prédicat dans la clause `HAVING` suit les mêmes règles de syntaxe qu'un prédicat figurant dans une clause `WHERE`. Cependant, il ne peut porter que sur les caractéristiques du groupe :

- soit sur les fonctions d'agrégation (mais pas nécessairement celles qui figurent dans la clause `WHERE`)
- soit sur des expressions (généralement des colonnes) figurant dans la clause `GROUP BY`.

EXEMPLE : Quels sont les nom et nombre moyen d'élèves de chaque professeur qui enseigne dans au moins deux classes :

```
SELECT p.nom, AVG(effectif)
FROM enseigne   JOIN classe ON num_classe = no  JOIN personne p ON num_prof = num_p
GROUP BY num_prof HAVING COUNT(DISTINCT num_classe) > 1;
```

1.3.3 Agrégation sur deux niveaux

La clause `GROUP BY` renvoyant plusieurs résultats, on peut appliquer une fonction d'agrégation sur une sélection obtenue avec un regroupement.

EXEMPLE : Quelle est la moyenne du nombre total d'élèves par professeur (ayant des élèves...).

```
SELECT AVG("somme") FROM
        ( SELECT SUM(effectif) AS "somme"
        FROM classe c JOIN enseigne en ON en.num_classe = c.no
        GROUP BY en.num_prof);
```

1.4 Bloc de qualification

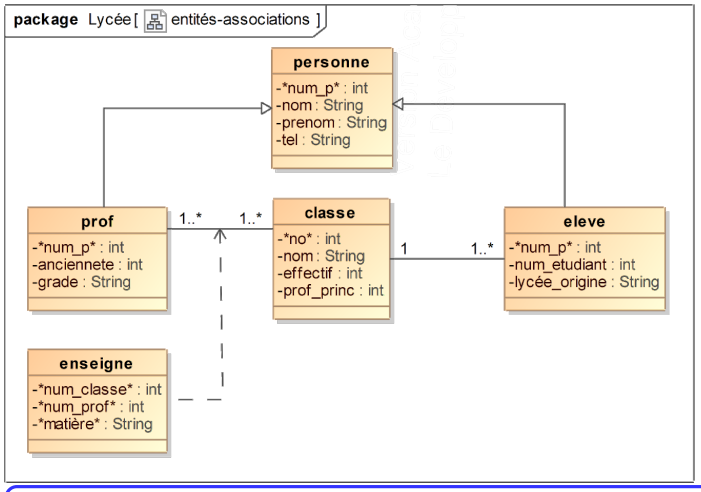
Après avoir vu les principales clauses d'une requête SQL, la structure de base est un bloc de qualification où la syntaxe impose l'ordre suivant :

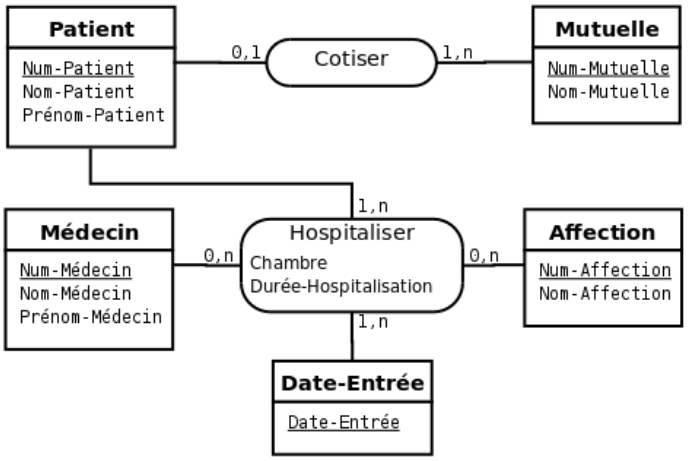
```
1 SELECT Ai,...,An          -- colonnes et agregations
2 FROM R                    -- relation
3 WHERE F                   -- assertion
4 GROUP BY A                -- regroupement
5 HAVING H                  -- assertion
6 ORDER BY T                -- tri
7 ;
```

2 Représentations graphiques

2.1 Modèle entités-associations

A l'aide d'un diagramme de classe d'UML, un schéma relationnel appelé diagramme entités-associations peut être représenté :





2.2 Éléments graphiques associés à l'algèbre relationnel

Pour raisonner sur le papier ou traduire visuellement une requête voici quelques éléments de langage graphique :