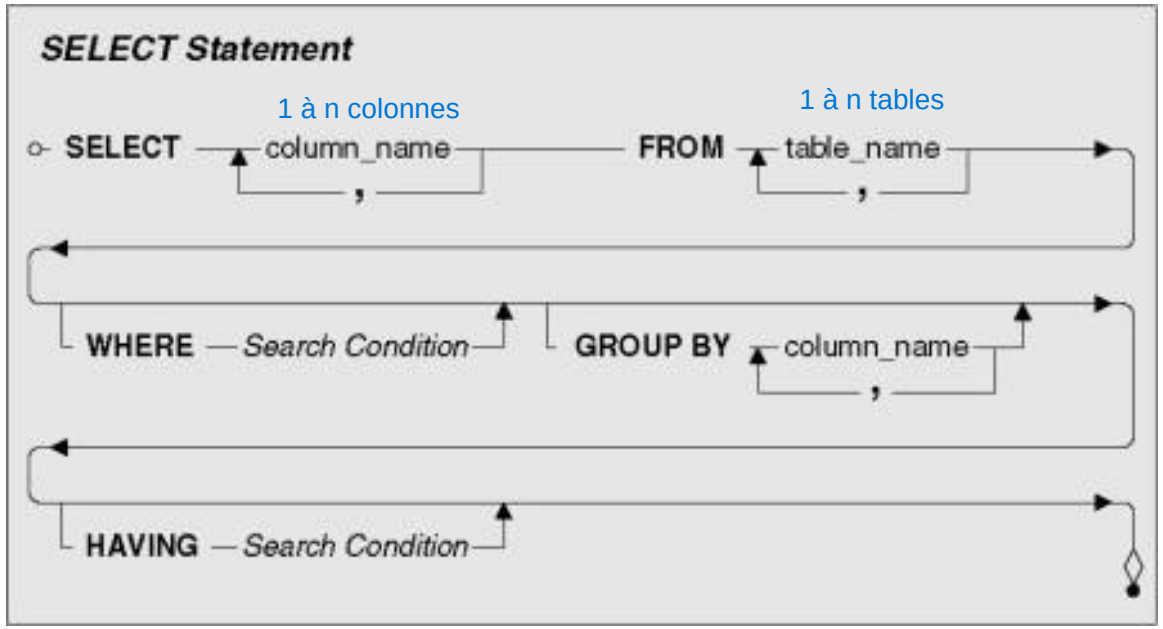
Base de données

Langage d'interrogation des données

Objectif

- L'extraction des données : syntaxe de base
- La projection et les calculs élémentaires
- La restriction
- Le calcul d'agrégat
- Le tri des résultats
- Mise en relation des ensembles : Le produit cartésien
- Mise en relation des ensembles : La jointure
- Mise en relation des ensembles : La jointure externe
- Le stockage des résultats intermédiaires

L'extraction des données : syntaxe de base



having: un peu équivalent à where mais appliqué au group by

La projection et les calculs élémentaires

- Sélection de colonnes :
  - Col1, col2, …
  - *
  - Table_name.col1, Table_name.col2, … S'il y a plusieurs tables avec des colonnes qui ont le même nom, on utilise table_name.col1...
  - Calcul élémentaire
  - Alias de colonne
  - Mots clés : **ALL** ou DISTINCT

Un Alias ajoute une colonne dans l'affichage du select (par rapport à la Table SQL)

DISTINCT: retire les doublons

on concatène: ici on doit utilise la fonction CONCAT
(le + ne fonctionne mas en MySQL

identite est un Alias

Ne le: alias de col

```
select codeEmp,
        upper(nom) + ' ' + prenom as identite,
        isnull(convert(varchar,dateNaissance,103), '<non renseignée>') [Ne le],
        convert(varchar,dateEmbauche,103) [Embauche le],
        SalaireAugmente = salaire * 1.1,
        Cout = (salaire * 1.1) - salaire
    from Employes;
```

on remplace les "vide" par non renseignée

valeur 103 (CONVERT): manière de formater les date: jj/mm/aaaa (code peu lisible)-
On préfère : CONVERT(varchar,dateNaissance,'YYYY-MM-DD HH:mm:ss')
ou ,'DD/MM YYYY HH:mm:ss' par exemple

La restriction

- Sélection des lignes :
  - Clause WHERE
  - Condition booléenne composée :
    - de colonnes, EX.: where(age<35)
    - de constantes,
    - de fonctions,
    - d'opérateurs de comparaison et logiques and, or, =,>,...

```
select * from Employes
    where dateNaissance is null or salaire = 0;
```

le "or" n'est pas exclusif (comme en math)

Le calcul d'agrégat

- Regroupement des lignes :
  - Fonctions d'agrégat
  - Clause GROUP BY
  - Clause HAVING

Le HAVING permet de mettre une condition qu'on ne pouvait pas mettre dans le where

codeEmp	Nom	codeService	codeChef
1	ALBERT	compta	3
2	DUPONT	inform	5
3	MARTIN	compta	5
4	DUPOND	inform	2
5	BOSS	dirgen	null

Projection et restriction

```
select codeService,
       COUNT(*) as NbEmployes,
       COUNT(codechef) as NbEmployesAvecChefs,
       SUM(salaire) as MasseSalariale,
       AVG(salaire) as Moyenne,
       MIN(salaire) as PlusBasSalaire,
       MAX(salaire) as PlusHautSalaire
from Employes
where UPPER(rtrim(codeService)) in ('RESHU','INFOR')
group by codeService
having SUM(salaire) > 5000;
```

Compte le nb d'employe dans chaque service

combien d'employés ont 1 chef?

somme salaire groupé par service

codeService
compta
compta
dirgen
inform
inform

Étiquette compta

Étiquette dirgen

Étiquette inform

Regroupement et calcul

group by: on doit grouper sur l'intégralité des lignes qui ne sont pas des fonctions d'agrégat

codeService=='reshu' ou 'infor'

codeService	NbEmployes	NbEmployesAvecChefs
Compta	2	2
Dirgen	1	0
Inform	2	2

Le GROUP BY doit contenir au minimum la liste des colonnes projetées.
Les colonnes de calcul d'agrégat n'apparaissent pas dans le GROUP BY.

Le tri des résultats

- Tri :
  - Clause ORDER BY
  - Options ASC ou DESC
  - Toujours la dernière clause du SELECT

```
select codeService,
       upper(nom) + ' ' + prenom as identite
  from Employes
  where UPPER(rtrim(codeService)) in ('RESHU','INFOR')
  order by codeService ASC, 2 DESC;
```

ASC: ascendant (croissant)
DESC: descendant (décroissant)

Tri à la fin (order by), pour qu'il se fasse sur une "base" soit la + petite possible

autant de tri de suite que l'on veut: order by codeService, salaire;
On peut faire "order by" sur une col qui n'est pas dans le select

Le tri a un coût. Il doit être réalisé sur le résultat final présenté à l'utilisateur.

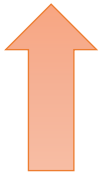
Rem.: SQL est très pratique pour faire du TRI.

Mise en relation des ensembles

- Le produit cartésien :
  - 2 syntaxes envisageables :
    - Syntaxe simplifiée
    - Syntaxe ANSI
  - Clause CROSS JOIN =prod cartesien
  - Alias de table

```
--syntaxe ANSI
select s.codeService, libelle, nom, prenom
    from [Services] as s cross join Employes e;
```

```
--syntaxe simplifiée
select s.codeService, libelle, nom, prenom
    from [Services] as s, Employes e;
```



Alias "s" sur le nom de table
"employe e" = "employe as e"

cross join: croise les services avec les employés
(pour être utile, il nécessite des restrictions)

Cross join: On évite car peu optimisé en terme de performance

Mise en relation des ensembles

- La jointure :
  - Produit cartésien + restriction
  - 2 syntaxes envisageables :
    - Syntaxe simplifiée
    - Syntaxe ANSI
  - Clause INNER JOIN … ON
  - Alias de table

La condition de jointure peut porter sur n'importe quelle colonne

```
--syntaxe ANSI
--les services non associés à au moins 1 employé
--ne figurent pas dans le résultat
select s.codeService, libelle, nom, prenom
    from [Services] s
                                --condition de jointure
        inner join Employes e on s.codeService = e.codeService;
```

```
--syntaxe simplifiée
--les services non associés à au moins 1 employé
--ne figurent pas dans le résultat
select s.codeService, libelle, nom, prenom
    from [Services] s, Employes e
    where       where remplace on
        --condition de jointure
        s.codeService = e.codeService
```

Pas très performant à cause du prod cartésien

Jointure

- Listez les employés ayant pris des congés en 2006 et à qui il restait plus de 10 jours de congés à prendre à la fin de l'année.

Faire un left join

Mise en relation des ensembles

- La jointure externe :
  - Clause {LEFT | RIGHT | FULL} OUTER JOIN … ON

Désigne la table qui a la priorité (à gauche ou à droite), presque toujours un LEFT JOIN (puisqu'un RIGHT est identique si on inverse l'ordre des tables)

Full join : mix entre left et right join (pas sûr que ça marche en MySQL

On peut faire des LEFT JOIN en cascade

"JOIN" tout court fonctionne
(cross join inutile)

```
--jointure externe : toutes les lignes de la table Services
-- + éventuellement les informations Employes si elles existent
select s.codeservice, libelle, COUNT(e.codeEmp) as NbEmployes
    from [Services] s left outer join Employes e on s.codeService = e.codeService
    group by s.codeservice, libelle
    order by 1;
```

Jointure: Partie la plus pénible sur les BDD

Jointure externe

- Listez le détail des congés mensuels pris par chaque employé en 2005. Les employés n'ayant pas pris de congés cette année là font partie du résultat.

Le stockage des résultats intermédiaires

- Requête de création de table :
  - Stocker temporairement les résultats intermédiaires.
  - Clause INTO
  - #Table_name : table temporaire locale
  - ##Table_name : table temporaire globale
  - Structure de la table identique à la projection.
  - Drop table

Accessible que par moi (ne marche pas sur MySQL)

Visible par tout le monde

Create temporary table...

!! Bcp de tables temporaires restent et polluent la BDD

```
select AVG(salaire) as moyenne
    into #T1    --table temporaire locale
    from Employes;
```

Chaque colonne projetée doit être nommée.

Le stockage des résultats intermédiaires

- Listez l'employé le mieux payé par service.

Base de données

Langage d'interrogation des données