IFT-6800, Automne 2016

Cours #6–Requêtes SQL

Louis Salvail
André-Aisenstadt, #3369
salvail@iro.umontreal.ca

SQL

- SQL: *Structured Query Language* est un langage informatique pour exploiter les bases de données relationnelles.
- En 1970, Edgar Frank Codd publia un article *"A Relational Model of Data for Large Shared Data Banks"* qui introduit les bases de données relationnelles fondées sur la logique mathématique du premier ordre: l'algèbre relationnelle.
- Reconnu rapidement comme un modèle intéressant pour faire des requêtes sur des bases de données.
- La première version commercialement disponible de SQL a été présentée en 1979 par *Relational Software inc*.

Ses parties

- La première partie est un *langage de manipulation des données*. Il permet de rechercher, d'ajouter, de modifier ou de supprimer des données dans la base de données.
- La seconde partie est un *langage de définition des données*. Il permet de créer et de modifier l'organisation des données dans la base de données.
- La troisième partie est un *langage de contrôle des transactions* qui permet d'initier et de terminer des transactions.
- La quatrième partie est un *langage de contrôle des données* qui permet d'autoriser et d'interdire l'accès à certaines données par certaines personnes.

Son utilisation

- On peut interagir avec le DBMS SQL avec une interface *ligne de commande*. On se connecte au DBMS (un serveur) et lui transmet des demandes en langage SQL.
- On peut écrire un programme dans un langage de programmation (comme Java avec JDBC) qui utilise *l'interface de programmation du DBMS SQL* pour lui transmettre des instructions SQL. Le programmeur ne fait pas de SQL à proprement dit, il utilise son langage de programmation pour acheminer les commandes SQL.
- On peut utiliser *la technique d'incorporation* qui incorpore des instructions en langage SQL dans un programme écrit dans un autre langage.
- Des *procédures écrites en SQL peuvent être stockées* dans la bases de données pour être exécutées par le DBMS SQL. Par exemple, des procédures qui s'exécutent automatiquement sur une modification du contenu de la base de données. Ceci se nomme *trigger (déclancheur)* en SQL.

Sur sa syntaxe

- Le langage SQL utilise une syntaxe qui ressemble à la syntaxe de phrases en anglais pour en faciliter l'apprentissage.
- Il s'agit d'un langage déclaratif: Un langage qui indique le résultat que l'on veut sans dire comment le trouver. Le DBMS est responsable pour produire le résultat de la meilleure façon possible. Il est équipé d'un *optimiseur* de requêtes qui trouve une bonne façon de produire le résultat.
- Dans ce cours, nous allons nous concentrer sur le langage de manipulation de données. Les instructions de ce type sont: `SELECT`, `UPDATE`, `INSERT` ou `DELETE`. Nous n'explorerons que l'instruction `SELECT`, car les autres sont des instructions pour modifier la base de données. Vous n'aurez pas les droits pour modifier la base de données sur le serveur.
- Les instructions qui permettent de manipuler l'organisation des données en SQL: `CREATE`, `ALTER` ou `DROP`. Nous en parlerons un peu plus loin.

L'organisation des données en SQL

- Vous pouvez vous connecter à un serveur mySQL avec:

  `mysql -h` serveur `-u` usager

  Si le serveur est sur votre machine alors `localhost` est indiqué après `-h`.

  Si le serveur est votre propre machine alors il y a de bonnes chances pour que `mysql` fonctionne sans argument.

- Une fois connecté à un serveur SQL, vous devez indiquer quelle base de données vous voulez utiliser:

  ```
  use Baseball;
  ```

  En autant que la base de données existe et qui vous y avez droit!

  indique à SQL d'utiliser la base de données `Baseball`.

- Vous pouvez créer une base de données avec:

  ```
  create database Baseball;
  ```

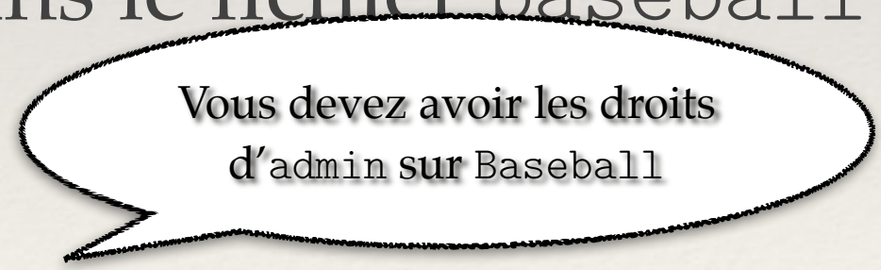
  La base de données `Baseball` est initialement vide.

Charger un nouvelle base de données

- En SQL, un fichier texte qui contient toutes les données d'une base de données peut être créé avec l'application (sous UNIX):

```
mysqldump -host 'hôte' -user 'nom' Baseball>baseball.sql
```

- Si vous voulez maintenant charger le base de données `Baseball` avec les données dans le fichier `baseball.sql` alors :



  - `mysql -h 'hôte' -u 'admin' Baseball<baseball.sql`

Consulter son organisation

- Si vous voulez voir l'organisation de la base de données que vous utilisez alors deux commandes s'offrent à vous:
  - `SHOW DATABASES;`
  - `SHOW TABLES;`
  - `DESCRIBE nom_table;`

Montre les bases de données auxquelles vous avez accès.

Montre les tables de la base de données courante.

Donne la description de la table *nom_table*.

Formulation des requêtes

- En SQL, les requêtes sont faites à l'aide d'une commande qui commence par le mot `SELECT`.
- Dans sa forme la plus simple, la commande `SELECT` a la forme suivante:

```
SELECT « quoi voir »
FROM   « dans quelles tables »
WHERE  « sur quelles conditions booléennes »
```

Exemple:Consultation d'une table

- Supposons que vous vouliez voir tous les salaires des joueurs des Expos de Montréal en 1992.
- Identifions ce que nous voulons voir…
- Identifions de quelle table l'information provient.
- Identifions quelle condition booléenne doit être satisfaite pour obtenir le résultat.

Salaries

```
+----------+------------+------+-----+---------+
| Field    | Type       | Null | Key | Default |
+----------+------------+------+-----+---------+
| yearID   | int(11)    | NO   | PRI | 0       |
| teamID   | varchar(3) | NO   | PRI |         |
| lgID     | varchar(2) | NO   | PRI |         |
| playerID | varchar(9) | NO   | PRI |         |
| salary   | double     | YES  |     | NULL    |
+----------+------------+------+-----+---------+
```

yearID=1992 ET teamID='MON'

La commande SQL résultante

SELECT *
retourne tous les attributs

```
SELECT playerID, salary
FROM salaries
WHERE (yearID=1992)AND(teamID='MON');
```

est vrai si yearID=1992

Les chaînes de caractères sont mises entre ' '

Les identificateurs d'équipes sont faits de 3 lettres

playerID	salary
aloumo01	110500
barbebr01	129000
bradlph01	200000
…	…

L'exécution de la commande

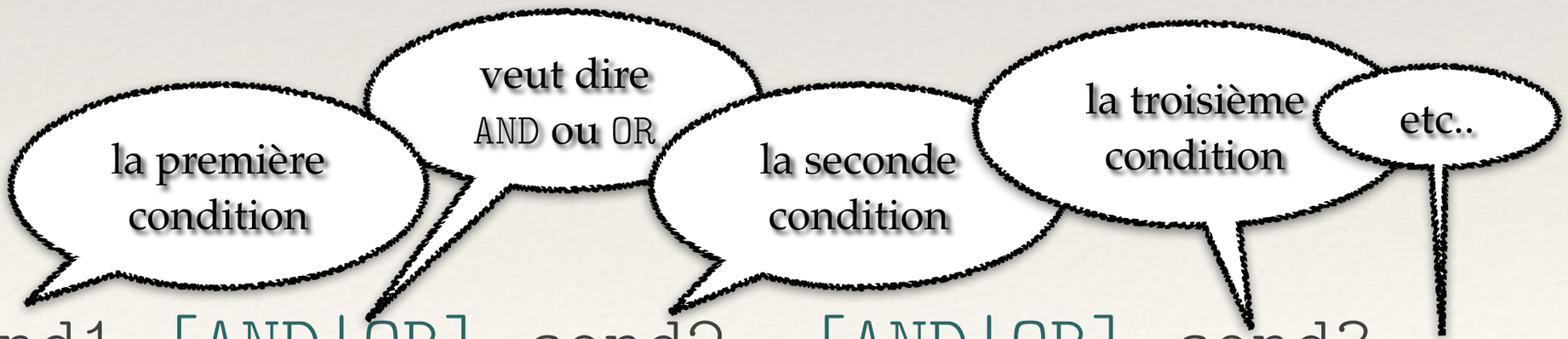
- Pour exécuter la commande précédente, le DBMS visite chaque ligne de la table `Salaries` qui est indiquée dans la clause `FROM`.
- Il conserve les lignes qui satisfont la condition booléenne de la clause `WHERE`.
- La table résultante est ensuite purgée des attributs qui ne sont pas dans la clause `SELECT` pour obtenir la table finale. La table finale est retournée à l'usager…

Les conditions de la clause `WHERE`

- La clause `WHERE` permet de filtrer les lignes de la tables résultante pour produire seulement celles qui nous intéressent.

- Les lignes qui demeurent dans la table finale sont celles qui satisfont la condition booléenne de la clause `WHERE`.

- Voyons le principal type de condition que nous pouvons exprimer:




```
WHERE cond1 [AND|OR] cond2  [AND|OR] cond3 ...
```

a=12
b=10
c=9.5

Exprimer une condition

a='abd'
b='abs'
c='abdabs'

- Supposons que a, b et c sont des attributs.
- Une condition est une expression booléenne sur la valeur d'attributs. Sa formulation dépend du type de d'attribut:
- Pour des attributs qui sont des nombres:
  - a=b, a<>b, a<b, a<=b, a>b, a>=b, 2*a/b>0, (a+b)>c.
- Pour des attributs qui sont des chaînes de caractères:
  - a=b, a||b=c, a<>b, a||b<>c, a<b, a<=b, a>b, a>=b, a>=b||c.

Exprimer une condition (II)

- Il y a d'autres conditions que nous pouvons exprimer sur des attributs formés d'une chaîne caractères permettant de tester la ressemblance entre attributs:

a contient 'Expo' plus un caractère de plus

`a LIKE 'Expo_', a LIKE 'Exp%', a NOT LIKE 'Expo_', a NOT LIKE 'Exp%'.`

a débute par 'Expo'

- Si les attributs sont de type temps ou date alors:

`a=b, a<>b , a<b, a<=b, a>=b, a>b.`

même date ou moment

date ou moment différent

a est avant b

a est au plus tard b

Tester si une valeur est `NULL`

- Vous pouvez également tester si une valeur est `NULL` dans la clause `WHERE`:
  - `WHERE (Master.deathYear <> NULL);`

Une autre requête sur une table

- Nous voulons les joueurs qui jouaient à New York (pour l'une ou l'autre des deux équipes) en 1985 pour un salaire moindre que 150 000$…

```
SELECT yearID,teamID,lgID
       playerID, salary
FROM salaries

WHERE (yearID=1985)AND
      (teamID LIKE 'NY_')AND
      (salary<150000);
```

yearID	teamID	lgID	playerID	salary
1985	NYA	AL	cowlejo01	120000
1985	NYA	AL	meachbo01	100000
1985	NYN	NL	gardero01	100000
1985	NYN	NL	johnsho01	115000
1985	NYN	NL	mcdowro01	60000
1985	NYN	NL	santara01	125000

La clause `ORDER BY`

- Lorsqu'il est souhaitable de trier les éléments générée par une requête, la clause `ORDER BY` peut être utilisée:
  - `ORDER BY attribut DESC` ou
  - `ORDER BY attribut ASC`
- Si l'attribut qui sert à trier est une valeur numérique alors l'ordre est l'ordre naturel.
- Même chose pour le temps est les dates…
- Si l'attribut est une chaîne de caractères alors l'ordre est celui du dictionnaire.

Exemple

- Supposons que nous voulons trier les joueurs en fonction de leur salaire du plus élevé au moins élevé dans la requête précédente:

```
SELECT playerID, salary

FROM salaries

WHERE (yearID=1985)AND
      (teamID LIKE 'NY_')AND
      (salary<150000)

ORDER BY salary DESC;
```

playerID	salary
santara01	125000
cowlejo01	120000
johnsho01	115000
meachbo01	100000
gardero01	100000
mcdowro01	60000

Agrégats

- Souvent, nous cherchons une valeur particulière associée à une table. Par exemple, le plus haut salaire dans le baseball majeur.
- Les agrégats permettent d'y parvenir. Ils sont appliqués aux éléments de la clause `SELECT`. Ils sont des fonctions appliquées à l'ensemble des attributs d'une table générée lors d'une requête juste avant de purger les colonnes qui ne sont pas indiquées dans la clause `SELECT`.
- Les agrégats habituels en SQL sont:
  - `AVG, MIN, MAX, COUNT, COUNT DISTINCT`.

Requêtes avec agrégats

- Quel était le salaire le plus élevé pour un joueur de baseball en 2010?
- Combien de joueurs gagnaient plus de 20 000000$ par année en 2009?
- Comment se compare le salaire maximal en 2010 avec le salaire moyen?

```
SELECT MAX(salary)
FROM Salaries
WHERE yearID=2010;
```

```
+-------------+
| max(salary) |
+-------------+
|    33000000 |
+-------------+
```

```
SELECT MAX(salary), AVG(salary)
FROM Salaries
WHERE yearID=2010;
```

```
+-------------+-------------------+
| MAX(salary) | AVG(salary)       |
+-------------+-------------------+
|    33000000 | 3278746.825301205 |
+-------------+-------------------+
```

```
SELECT COUNT(playerID)
FROM Salaries
WHERE yearID=2009 and salary>20000000;
```

```
+---------------+
| count(salary) |
+---------------+
|             4 |
+---------------+
```

Un autre exemple

- Supposons que vous voulez savoir le nombre total de personnes différentes qui ont déjà possédé une équipe de baseball et qui ont au moins une saison avec plus de victoires que de défaites. Comment faire?
- Toutes ces informations sont dans la table Managers.

```
SELECT COUNT(playerID)
FROM Managers
WHERE W>L;
```

```
+-----------------+
| COUNT(playerID) |
+-----------------+
|            1584 |
+-----------------+
```

```
SELECT COUNT(DISTINCT playerID)
FROM Managers
WHERE W>L;
```

```
+--------------------------+
| COUNT(DISTINCT playerID) |
+--------------------------+
|                      382 |
+--------------------------+
```

Requêtes qui combinent l'info de plusieurs tables

- La requête précédente ne nécessitait la consultation que d'une seule table. Ce n'est évidemment pas le cas pour toutes les requêtes utiles.
- Considérez la requête qui vise à informer sur les prénoms et noms de famille des lanceurs qui ont eu plus de 20 victoires en 2011.
- Quelles tables doivent être consultées pour accéder à ces informations?

Pitching

Field	Type	Null	Key	Default	Extra
playerID	varchar(9)	NO	PRI		
yearID	int(11)	NO	PRI	0	
stint	int(11)	NO	PRI	0	
teamID	varchar(3)	YES		NULL	
lgID	varchar(2)	YES		NULL	
W	int(11)	YES		NULL	
L	int(11)	YES		NULL	
G	int(11)	YES		NULL	
GS	int(11)	YES		NULL	
CG	int(11)	YES		NULL	
SHO	int(11)	YES		NULL	
SV	int(11)	YES		NULL	
IPouts	int(11)	YES		NULL	
H	int(11)	YES		NULL	
ER	int(11)	YES		NULL	
HR	int(11)	YES		NULL	
BB	int(11)	YES		NULL	
SO	int(11)	YES		NULL	
BAOpp	double	YES		NULL	
ERA	double	YES		NULL	
IBB	int(11)	YES		NULL	
WP	int(11)	YES		NULL	
HBP	int(11)	YES		NULL	
BK	int(11)	YES		NULL	
BFP	int(11)	YES		NULL	
GF	int(11)	YES		NULL	
R	int(11)	YES		NULL	
SH	int(11)	YES		NULL	
SF	int(11)	YES		NULL	

Cherchons en premier lieu où trouver le nombre de victoires des lanceurs pour une année donnée…

Cherchons où trouver les noms et prénoms des lanceurs…

Nous avons donc besoin de consulter Pitching et Master pour trouver les noms des joueurs qui ont plus de 20 victoires en 2011

La requête

```
SELECT master.nameFirst, master.nameLast,
       pitching.W

FROM   pitching, master

WHERE (pitching.yearID=2011)AND
       (pitching.W>20)AND
       (master.playerID=pitching.playerID);
```

nameFirst	nameLast	W
Ian	Kennedy	21
Clayton	Kershaw	21
Justin	Verlander	24

Exécution d'une requête multi-table

```
SELECT *
FROM  tA, tB
WHERE tA.a=tB.u AND tB.v>100;
```

tA

a	b
1	A
2	B
3	B
4	A

tB

u	v
1	100
2	102
1	104
2	105

tA.a	tA.b	tB.u	tB.v
1	A	1	104
2	B	2	102
2	B	2	105

Informations supplémentaires et formatage

- Supposons que vous voulez également produire avec les lanceurs de plus de 20 victoires une indication de leur taux de succès:

```
SELECT  master.nameFirst as prénom,
        master.nameLast as nom_de_famille,
        pitching.W as Gains,
        pitching.W/(pitching.W+pitching.L) as Moy
FROM    pitching, master
WHERE (pitching.yearID=2011)AND
      (pitching.W>20)AND
      (master.playerID=pitching.playerID);
```

Résultat

```
+---------+----------------+-------+--------+
| prénom  | nom_de_famille | Gains | Moy    |
+---------+----------------+-------+--------+
| Ian     | Kennedy        |    21 | 0.8400 |
| Clayton | Kershaw        |    21 | 0.8077 |
| Justin  | Verlander      |    24 | 0.8276 |
+---------+----------------+-------+--------+
```

La clause `Group by`

- Il est dans certains cas utile de regrouper certaines lignes d'une table en fonction de la valeur d'un attribut.
- La clause `GROUP BY` permet de grouper les lignes d'une table qui satisfont la clause `WHERE` en fonction de la valeur d'un ou plusieurs attributs.
- Un exemple est le suivant:
  - Nous voulons avoir une table de toutes les équipes avec leur fiche à vie. Comment faire ceci?
  - Notez en premier lieu que ces informations sont toutes dans la table `Teams`.
  - Nous allons devoir grouper les équipes par groupe contenant toutes leurs éditions. Leur fiche à vie pourra alors être déterminée par un agrégat…

Exemple

- Écrivons une requête qui donne, pour chaque équipe, le nombre de victoires, le nombre de défaites, la moyenne durant toute son histoire:

```
SELECT teamID, SUM(W) as Gains,
       SUM(L) as Défaites,
       SUM(W)/(SUM(W)+SUM(L)) as Moy
FROM   Teams

GROUP BY teamID

ORDER BY Moy DESC;
```

Résultat

teamID	Gains	Défaites	Moy
SLU	94	19	0.8319
BS1	225	60	0.7895
BS2	93	42	0.6889
CH1	19	9	0.6786
MLU	8	4	0.6667
PH1	165	86	0.6574
CNU	69	36	0.6571
SL4	781	432	0.6439
BSP	81	48	0.6279
HAR	78	48	0.6190
PRO	438	278	0.6117
BLN	644	447	0.5903
SL3	73	51	0.5887
ML3	21	15	0.5833
CN2	549	396	0.5810
BRP	76	56	0.5758
IND	88	65	0.5752
SL2	39	29	0.5735
PH2	102	77	0.5698
NYA	9913	7495	0.5695
CHF	173	133	0.5654
NYP	74	57	0.5649
ML1	1146	890	0.5629
NY1	6067	4898	0.5533
NY2	151	122	0.5531

La clause `HAVING`

- La clause `HAVING` permet de grouper les résultats qui satisfont une condition et d'en retenir que des attributs agrégats.
- La condition doit porter sur les attributs agrégats ou sur l'attribut qui détermine le groupe.
- Essayons de grouper les saisons des équipes qui ont joué au moins 10000 parties dans leur histoire dans la requête précédente.
- De plus, restreignons notre classement aux équipes de la ligue nationale…
- Et écrivons le nom complet des équipes au lieu de leur `teamID`.

La requête résultante

```
SELECT name, SUM(W) as Gains,
       SUM(L) as Défaites,
       SUM(W)/(SUM(W)+SUM(L)) as Moy
FROM   Teams
WHERE  lgID = 'NL'
GROUP BY teamID
HAVING Gains+Défaites>=10000
ORDER BY Moy DESC;
```

ATTENTION: Ceci n'est peut être pas le résultat visé! Je triche, nous verrons pourquoi.

Résultat

Les vieux noms sont ceux indiqués ici parce que les noms changent et name n'est pas un agrégat du groupe! Un nom arbitraire du passé est affiché…

name	Gains	Défaites	Moy
New York Gothams	6067	4898	0.5533
Brooklyn Bridegrooms	5214	4926	0.5142
St. Louis Browns	9690	9269	0.5111
Chicago White Stockings	10511	10065	0.5108
Pittsburg Alleghenys	9907	9703	0.5052
Cincinnati Reds	9708	9529	0.5047
Boston Red Caps	5118	5598	0.4776
Philadelphia Quakers	9464	10551	0.4728

Un raffinement de plus

- Supposons que nous voulons également indiquer pour les équipes retournées par la requête précédente si elles sont toujours actives…
- Il s'agit simplement de modifier la clause `SELECT` de la dernière requête:

Cet
attribut est un agrégat booléen

```
SELECT name, Max(yearId)>=2013 as Active,
       SUM(W) as Gains,
       SUM(L) as Défaites,
       SUM(W)/(SUM(W)+SUM(L)) as Moy
```

Résultat

name	Active	Gains	Défaites	Moy
New York Gothams	0	6067	4898	0.5533
Brooklyn Bridegrooms	0	5214	4926	0.5142
St. Louis Browns	1	9690	9269	0.5111
Chicago White Stockings	1	10511	10065	0.5108
Pittsburg Alleghenys	1	9907	9703	0.5052
Cincinnati Reds	1	9708	9529	0.5047
Boston Red Caps	0	5118	5598	0.4776
Philadelphia Quakers	1	9464	10551	0.4728

0: FAUX

1: VRAI

Nous avons toujours le problème des noms (anciens) que nous réglerons plus tard.

La masse salariale des équipes

- Pour terminer, voyons une requête qui retourne la masse salariale des équipes de la ligue nationale pour l'année 2010:

```
SELECT name,
       SUM(salary) as MasseSalariale,
FROM   Teams, Salaries
WHERE  Teams.teamID=Salaries.teamID AND
       Teams.lgID = 'NL' AND
       Teams.yearID=2010 AND
       Salaries.yearID=2010
```

La requête

```
SELECT name, Teams.yearID as Année,
       SUM(salary) as MasseSalariale,

FROM   Teams, Salaries

WHERE Teams.teamID=Salaries.teamID AND
      Teams.lgID='NL' AND
      Teams.yearID=2010 AND
      Salaries.yearID=2010

GROUP BY teamID

ORDER BY MasseSalariale DESC;
```

Nous montrerons les noms de 2010 pour chaque teamID.

Résultat

Les noms sont maintenant les noms courants!

Les noms sont les noms courants, car ce sont les noms de 2010!

name	Année	MasseSalariale
Chicago Cubs	2010	146609000
Philadelphia Phillies	2010	141928379
New York Mets	2010	134422942
San Francisco Giants	2010	98641333
Los Angeles Dodgers	2010	95358016
St. Louis Cardinals	2010	93540751
Houston Astros	2010	92355500
Atlanta Braves	2010	84423666
Colorado Rockies	2010	84227000
Milwaukee Brewers	2010	81108278
Cincinnati Reds	2010	71761542
Washington Nationals	2010	61400000
Arizona Diamondbacks	2010	60718166
Florida Marlins	2010	57029719
San Diego Padres	2010	37799300
Pittsburgh Pirates	2010	34943000

Retour sur le problème des noms courants

- La requête qui retourne la fiche des équipes de plus de 10000 rencontres en indiquant leur nom courant…

```
SELECT tb.name, SUM(W) as Gains,
       SUM(L) as Défaites,
       SUM(W)/(SUM(W)+SUM(L)) as Moy
FROM   Teams as ta, Teams as tb
WHERE  ta.lgID = 'NL' AND
       ta.teamID=tb.teamID AND tb.yearID=2012
GROUP BY tb.name
HAVING Gains+Défaites>=10000
ORDER BY Moy DESC;
```

Sous-requêtes

- La requête utilise `tb` seulement pour ranger les noms (`name`) courants en 2012 avec le `teamID`.
- Au lieu de réduire la table `Teams.tb` dans la clause `WHERE`, on peut le faire dans la clause `FROM` en utilisant une sous-requête:

```
SELECT tb.name, SUM(W) as Gains, SUM(L) as Défaites,
       SUM(W)/(SUM(W)+SUM(L)) as Moy
FROM   Teams, (SELECT name, teamID
               FROM Teams
               WHERE lgID='NL' AND yearID=2012) as tb
WHERE  Teams.teamID=tb.teamID
GROUP BY tb.name
HAVING Gains+Défaites > 10000 ORDER BY Moy DESC;
```

Joindre des tables explicitement

- Dans la requête précédente, nous pourrions également joindre les tables Teams et tb en fonction de leur teamID:

```
SELECT tb.name, SUM(W) as Gains,
       SUM(L) as Défaites,
       SUM(W)/(SUM(W)+SUM(L)) as Moy
FROM   Teams INNER JOIN
       (SELECT name, teamID FROM Teams
        WHERE lgID='NL' AND yearID=2012) as tb
       ON Teams.teamID=tb.teamID
GROUP BY name
HAVING Gains+Défaites > 10000 ORDER BY Moy DESC;
```

La clause WHERE n'est plus nécessaire, car la condition est exprimée dans la clause ON de INNER JOIN

Générer des vues

- Vous pouvez ranger le résultat d'une requête sur le serveur de base de données pour son utilisation dans une requête future, par exemple.
- Nous appelons vue (`VIEW`) une telle table qui est générée par une requête et rangée (temporairement) sur le serveur:
  - `CREATE VIEW Teams2010 AS`

    `SELECT * FROM Teams WHERE yearID=2010;`
- Cette commande produit la table `Teams2010` sur le serveur. Cette table peut être utilisée dans une requête comme les autres. Elles sont toujours remises à jour lorsque la table `Teams` est modifiée, par exemple.

Conclusion

- SQL permet de faire des requêtes plus complexes que celles que nous venons de voir:
  - Les opérations de jointures de tables sont plus générales que le `INNER JOINT`,
  - Les tests dans la clause `WHERE` peuvent avoir des test qui contiennent des sous requêtes SQL,
  - Les agrégats peuvent être utilisés de façon plus générales, etc…
- Nous avons cependant vu les outils nécessaires à la très grande majorité des requêtes.
- SQL est très puissant pour formuler des requêtes sur des bases des données. Cependant, il n'a pas la puissance du langage machine de la XTOY puisque peu importe comment vous procédez, SQL ne boucle jamais à l'infini!!!!