Instruction SELECT

L'instruction SELECT est la base du LMD, elle permet de renvoyer une table contenant les données correspondantes aux critères qu'elle contient.

Seules les clauses "SELECT" et "FROM" sont obligatoires. La forme générale d'une instruction SELECT (on parle également de *phrase SELECT* ou *requête*) est :

```
SELECT [ALL] | [DISTINCT] * | <liste de champs ou d'instructions d'agrégation>
FROM <liste de tables>
WHERE <condition>
GROUP BY <champs de regroupement>
HAVING <condition>
ORDER BY <champs de tri> [DESC] | [ASC]
```

Une autre forme est

```
SELECT [ALL] | [DISTINCT] * | <liste de champs ou d'instructions d'agrégation>
FROM <table de base>
<liste de jointures>
GROUP BY <champs de regroupement>
HAVING <condition>
ORDER BY <champs de tri> [DESC] | [ASC]
```

Détails des clauses :

SELECT

La clause SELECT permet de spécifier les informations qu'on veut récupérer. Elle contient des champs provenant de tables spécifiées dans la clause FROM, ainsi que des instructions d'agrégation portant sur ces champs.

Le nom des champs ne doit pas être équivoque, ce qui veut dire que si des champs de tables différentes ont le même nom, les champs doivent être préfixés par le nom de la table : `nom_de_table.nom_de_champs`

Les noms des champs sont séparés par des virgules. Si la requête comporte la clause DISTINCT, les doublons (lignes de la table de résultat ayant exactement les mêmes valeurs dans tous les champs) seront éliminés, alors qu'avec la clause ALL, tous les résultats sont renvoyés.

SELECT * permet de renvoyer tous les champs de toutes les tables spécifiées dans FROM.

Instructions d'agrégation

Les instructions d'agrégation permettent des opérations comme le comptage ou les sommes.

Les différentes instructions d'agrégation sont :

- AVG(<champs>)
- COUNT(*)
- MAX (<champs>)
- MIN (<champs>)
- SUM (<champs>)

FROM

Clause obligatoire qui détermine sur quelles tables l'on fait la requête. Les noms des tables sont séparés par des virgules.

Exemple :

```
SELECT nom, prenom
FROM table_president ;
```

Résultat :

nom	prenom
Chirac	Jacques
Kaczynski	Lech
Napolitano	Giorgio
Bachelet	Michelle

Pour voir l'utilisation avec plusieurs tables voir plus bas le chapitre **jointure**.

Si vous voulez récupérer tous les enregistrements de la table, il suffit de remplacer le nom des champs par une "étoile".

Exemple :

```
SELECT *
FROM table_president;
```

Ou

```
SELECT table_president.*
FROM table_president ;
```

Résultat :

genre	nom	prenom	pays
m	Chirac	Jacques	France
m	Kaczynski	Lech	Pologne
m	Napolitano	Giogio	Italie
f	Bachelet	Michelle	Chili

Vous pouvez renommer les champs dans le résultat de la sélection.

Exemple :

```
SELECT nom AS "name", prenom AS "firstname"
FROM table_president ;
```

Résultat :

name	firstname
Chirac	Jacques
Kaczynski	Lech
Napolitano	Giorgio
Bachelet	Michelle

Vous pouvez adapter le résultat à certains critères.

Exemple :

```
SELECT
  CASE genre
    WHEN "m" THEN "Monsieur le président"
    WHEN "f" THEN "Madame la présidente"
    ELSE "Erreur"
  END AS "salutation", nom, prenom
FROM table_president ;
```

Résultat :

salutation	nom	prenom
Monsieur le président	Chirac	Jacques
Monsieur le président	Kaczynski	Lech
Monsieur le président	Napolitano	Giorgio
Madame la présidente	Bachelet	Michelle

Vous pouvez utiliser le "plus" pour concaténer les chaînes de caractères.

Exemple :

```
SELECT prenom + " " + nom + "(" + pays + ")" AS "Nom et pays"
FROM table_president ;
```

Résultat :

Nom et pays
Chirac Jacques (France)
Kaczynski Lech (Pologne)
Napolitano Giorgio (Italie)
Bachelet Michelle (Chili)

Vous pouvez aussi faire des opérations arithmétiques (+, -, *, /) sur les champs.

Exemple :

```
SELECT prix, taux_tva, remise, prix + (prix /100 * taux_tva) - remise AS "total"
FROM table_commande ;
```

Résultat :

prix	taux_tva	remise	total
50	10	7	48
200	25	0	250

Alias

Vous pouvez utiliser un alias, c'est-à-dire un renommage provisoire, pour le nom de la table. Il suit le nom de la table dont il est séparé par une espace. Ceci n'a aucun intérêt à ce stade, mais ça permettra d'abréger la rédaction de vos requêtes quand vous écrirez des conditions, et ce sera très utile pour écrire une auto-jointure. *Exemple :*

```
SELECT *
FROM table_president pres ;
```

Résultat :

genre	nom	prenom	pays
m	Chirac	Jacques	France
m	Kaczynski	Lech	Pologne
m	Napolitano	Giogio	Italie
f	Bachelet	Michelle	Chili

Plusieurs tables

Si vous invoquez plusieurs tables à la fois sans utiliser de clause "WHERE" ni de jointure (un peu de patience, voir ci-dessous), vous obtenez alors une "jointure croisée" (CROSS JOIN) ou "produit cartésien", qui devrait être expliqué plus bas un jour. Les noms des tables sont alors séparés par des virgules. *Exemple :*

```
SELECT *
FROM table_president, table_premier_ministre ;
```

WHERE

Le mot clef "`WHERE`" permet de mettre des conditions à la requête, qui permettent de sélectionner certaines lignes. Les conditions sont des comparaisons entre les champs.

Elément	Description
>	Plus grand que

>=	Plus grand ou égal à
<	Plus petit que
<=	Plus petit ou égal à
=	Égal à
<>	Différent de

Exemple :

```
SELECT *
FROM table_eleve ;
```

Résultat :

nom	age
Paul	8
Jean	7
Jacques	8
Sylvie	9
Steve	8
Julie	7

Pour avoir tous les élèves qui ont 8 ans et plus l'on fait ces requêtes.

```
SELECT *
FROM table_eleve
WHERE age >= 8 ;
```

ou

```
SELECT *
FROM table_eleve
WHERE age > 7 ;
```

Pour avoir les élèves qui ont exactement 8 ans.

```
SELECT *
FROM table_eleve
WHERE age = 8 ;
```

Pour avoir tous les élèves qui n'ont pas 9 ans.

```
SELECT *
FROM table_eleve
WHERE age <> 9 ;
```

Pour avoir les élèves qui se nomment Jean.

```
SELECT *
FROM table_eleve
WHERE nom = "Jean" ;
```

Attention, je dois écrire Jean correctement en faisant attention aux espaces. "Jean " ou "jean" ne fonctionnera pas (espace, minuscule).

Pour la comparaison de chaîne de caractères regarder le mot clef "`LIKE`".

AND et OR

Vous pouvez chaîner des conditions avec les mots clef "`AND`" (et), "`OR`" (ou inclusif, c'est-à-dire que si les 2 conditions sont vraies ensemble, la condition est vérifiée).

Sélectionner les élèves qui ont 7 ou 9 ans et dont le nom commence par J

Exemple :

```
SELECT *
FROM table_eleve
WHERE (age = 7
  OR age = 9)
  AND nom Like "J%" ;
```

Sélectionner les élèves qui ont 7 ans, ou qui ont un nom qui commence la lettre par J, ou les deux.

Exemple :

```
SELECT *
FROM table_eleve
WHERE age = 7 OR nom Like "J%" ;
```

IN et NOT IN

Vous pouvez aussi faire la comparaison avec une liste grâce à "`IN`" et "`NOT IN`".

Sélectionner uniquement les élèves qui ont 7 ans et ceux qui ont 9 ans.

Exemple :

```
SELECT *
FROM table_eleve
WHERE age IN ( 7 , 9 );
```

Résultat :

nom	age
Jean	7
Sylvie	9
Julie	7

Pour avoir les élèves qui ne se nomment pas Jean ou Steve ou Julie.

Exemple :

```
SELECT *
FROM table_eleve
WHERE nom NOT IN ("Jean" , "Steve" , "Julie");
```

Résultat :

nom	age
Paul	8
Jacques	8
Sylvie	9

BETWEEN

Avec "`BETWEEN`" vous pouvez sélectionner des valeurs dans un intervalle.

Pour sélectionner les élèves qui ont entre 6 et 8 ans :

Exemple :

```
SELECT *
FROM table_eleve
WHERE age BETWEEN 6 AND 8 ;
```

Et avec le "`NOT BETWEEN`" vous pouvez sélectionner les élèves qui ne sont pas dans la tranche d'âge 6-8 ans.

Exemple :

```
SELECT *
FROM table_eleve
WHERE age NOT BETWEEN 6 AND 8 ;
```

LIKE

Condition sur une chaîne de caractères.

- _ : Joker qui remplace exactement un caractère.
- % : Joker qui remplace une suite de caractère.

Critère : "J_l%"
Mots correspondant : Jules, Jel, Julie, Julien, Jolien

Pour sélectionner des courriels qui semblent valides.

Exemple :

```
SELECT *
FROM table_client
```

```
WHERE email LIKE "%@%.%" ;
```

IS NULL

`IS NULL` permet de sélectionner les valeurs nulles.

Exemple :

```
SELECT *
FROM table_client
WHERE adresse IS NULL ;
```

Le résultat sera l'ensemble des enregistrements de la table_client où les adresses sont nulles.

`IS NOT NULL` permet de sélectionner uniquement ceux qui n'ont pas la valeur nulle.

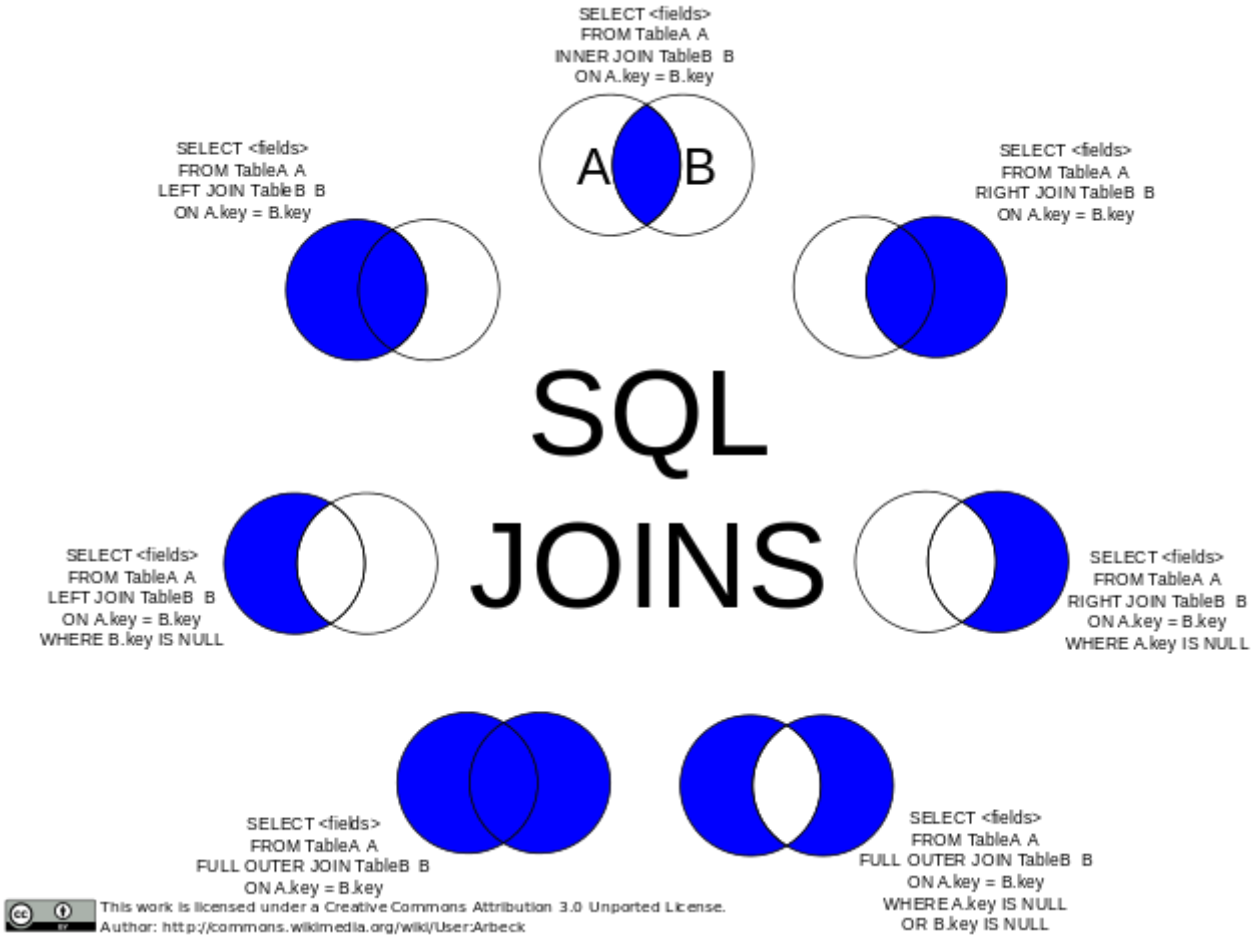
Exemple :

```
SELECT *
FROM table_client
WHERE adresse IS NOT NULL ;
```

Cette fois nous avons l'ensemble des enregistrements de la table_client qui ont une adresse.

Jointures

Les jointures permettent d'assembler les champs de différentes tables.



Signification des différentes jointure en SQL

Jointure interne

`INNER JOIN`

Exemple

Nous avons une table "eleves" d'élèves :

id_eleve	nom	age
1	Paul	8
2	Jean	7
3	Jacques	8
4	Sylvie	9
5	Steve	8
6	Julie	7

une table "branches" de branches :

id_branche	nom
1	Francais

2	Histoire
3	Math

et une table "notes" de notes :

id_note	id_eleve	id_branche	note
1	1	1	8
2	2	1	10
3	4	1	10
4	5	1	8
5	6	1	4

Pour afficher chaque note avec l'élève et la branche correspondante, je dois faire une jointure.

```
SELECT notes.id_note, eleves.nom as eleve, branches.nom as branche, notes.note
FROM eleves, branches, notes
WHERE eleves.id_eleve = notes.id_eleve
  AND branches.id_branche = notes.id_branche ;
```

ou

```
SELECT notes.id_note, eleves.nom as eleve, branches.nom as branche, notes.note
FROM notes INNER JOIN eleves ON notes.id_eleve = eleves.id_eleve
  INNER JOIN branches ON notes.id_branche = branches.id_branche ;
```

id_note	eleve	branche	note
1	Paul	Francais	8
2	Jean	Francais	10
3	Sylvie	Francais	10
4	Stève	Francais	8
5	Julie	Francais	4

Jointure externe

Dans le cas ci-dessus nous ne pouvons pas voir l'élève Jacques (3) qui n'a pas participé à l'examen de français. La jointure externe permet de ne pas perdre d'information en conservant soit toutes les lignes de la table de gauche (LEFT OUTER JOIN) soit à droite.

LEFT OUTER JOIN

Ici, en conservant toutes les lignes de la table eleves :

```
SELECT notes.id_note, eleves.nom as eleve, branches.nom as branche, notes.note
FROM eleves LEFT OUTER JOIN notes ON notes.id_eleve = eleves.id_eleve
  INNER JOIN branches ON notes.id_branche = branches.id_ branche ;
```

RIGHT OUTER JOIN

Pour conserver toutes les lignes des deux tables. ***FULL OUTER JOIN***

Auto jointures

ORDER BY

Cette clause permet d'ordonner les résultats d'une requête. On peut les ordonner de deux manières : ascendante (ou dans l'ordre croissant) avec le mot clé ASC ou descendant (décroissant) avec le mot clé DESC.

Reprenons notre modèle logique :

élèves(**id_élève**, nom_élève, prénom_élève, adresse_élève, cp_élève, ville_élève, **# id_classe**)

Si l'on désire afficher les élèves par ordre alphabétique croissant sur le nom, on fait :

```
SELECT *
FROM élèves
ORDER BY nom_élève ASC;
```

La requête ci-dessous est équivalente :

```
SELECT *
FROM élèves
ORDER BY nom_élève
```

La plupart des SGBD trient par défaut de manière croissante.

Nous pouvons préciser le tri. Par exemple nous voulons trier sur le nom, puis le prénom puis l'adresse de manière décroissante.

```
SELECT *
FROM élèves
ORDER BY (nom_élève, prénom_élève, adresse_élève) DESC;
```

GROUP BY

L'instruction GROUP BY en sql permet de grouper un ensemble d'enregistrements issus d'une requête, de façon à ce que chaque groupe ainsi créé ne soit représenté que par une seule ligne.

Prenons par exemple le modèle logique suivant à 3 tables :

élèves(**id_élève**, nom_élève, prénom_élève, adresse_éléve, cp_élève, ville_élève, **# id_classe**)

Classes(**id_classe**, nom_classe, **# id_section**)

section(**id_section**, nom_section)

Lorsque l'on veut afficher la classe, la section de chaque élève on procède comme suit :

```
SELECT
  nom_élève    AS "nom",
  prenom_élève AS "prenom",
  nom_classe   AS "classe",
  nom_section  AS "section"
FROM
  élèves  e,
  classes c,
  section s
WHERE
  e.id_classe=c.id_classe   AND
  c.id_section=s.id_section
```

Si nous voulons obtenir les effectifs de chaque classe nous faisons :

```
SELECT id_classe, Count(*)
FROM élèves
GROUP BY id_classe
```

Remarque :

- La présence de la clause GROUP BY est nécessaire dès que la requête comporte simultanément dans la clause de sélection (SELECT) le filtre (WHERE), des jointures, des colonnes de tables hors calcul et des calculs d'agrégation.
- Dans ce cas, toutes les colonnes représentées hors des calculs d'agrégation doivent figurer dans la clause GROUP BY.

HAVING

Cette clause est liée à la clause GROUP BY. Elle permet de préciser une sélection. Lorsque l'on applique GROUP BY, on effectue une réunion. HAVING va nous permettre d'émettre une condition sur cette réunion. La clause HAVING est largement utilisée avec les fonctions d'agrégation.

Reprenons par exemple le modèle logique de GROUP BY ci-dessus. Si cette fois nous voulons filtrer les effectifs et n'afficher que les classes qui ont un effectif inférieur ou égal à 20 élèves, nous faisons :

```
SELECT id_classe, Count(*)
FROM élèves
GROUP BY id_classe
HAVING Count(*) <= 20
```

Attention la clause HAVING doit impérativement précéder la clause ORDER BY.

Exemple :

```
SELECT id_classe AS "identifiant", nom_classe AS "nom", Count(*) AS "effectif"
FROM classes c, élèves e
WHERE e.id_classe=c.id_classe
GROUP BY (identifiant, nom)
HAVING Count(*) BETWEEN 15 AND 30
ORDER BY nom ASC
```

Instruction INSERT

Cette requête sert à insérer des enregistrements dans les tables d'une base de données.

La syntaxe de la commande est la suivante (cas le plus simple, la table étant déjà créée) :

INSERT INTO nom_table VALUES ("Paul","Jean","Isabelle","Marie");

Dans un cas plus général, reprenons notre exemple :

élèves(**id_élève**, nom_élève, prénom_élève, adresse_éléve, cp_élève, ville_élève, **# id_classe**)

Pour ajouter un enregistrement à cette table, nous faisons :

```
  INSERT INTO élèves (id_élève, nom_élève, prénom_élève, adresse_éléve, cp_élève, ville_élève,
 id_classe)
    VALUES ('1','brindzingue','toto','trifoullie les oyes','66000','un coin perdu','12');
```

Remarque : Si l'on ajoute tous les champs de l'enregistrement, il n'est pas utile de préciser le nom des champs (ex: nom_élèves, ville_élève, ...), il suffit de saisir, dans l'ordre des champs de la base de données (de la gauche vers la droite) les valeurs associées aux enregistrements.

L'exemple suivant est donc equivalent à l'exemple précédent :

```
  INSERT INTO élèves
    VALUES ('1','brindzingue','toto','trifoullie les oyes','66000','un coin perdu','12');
```

Pour n'ajouter que quelques données (données obligatoires par exemple, c'est-à-dire, celles qui ne doivent pas être nulles dans un enregistrement) on peut opèrer comme suit :

En imaginant que les données adresse_éléve, cp_élève et ville_élève soient optionnelles dans la base de données :

```
  INSERT INTO élèves (id_élève, nom_élève, prénom_élève, id_classe)
    VALUES ('1','brindzingue','toto','12');
```

est une requête valide.

Instruction UPDATE

Cette requête sert à mettre à jour des enregistrements dans les tables d'une base de données. La syntaxe de la commande est la suivante :

```
  UPDATE <nom_de_la_table> SET
    (<colonne_1>=<nouvelle_valeur> , <colonne_2>=<nouvelle_valeur>, …)
  [ WHERE <condition> ];
```

Reprenons notre exemple :

élèves(id_élève, nom_élève, prénom_élève, adresse_éléve, cp_élève, ville_élève, # id_classe)

Actuellement la table ressemble à cela :

id_élève	nom_élève	prénom_élève	adresse_éléve	cp_élève	ville_élève	# id_classe
1	brindzingue	toto	trifoullie les oyes	66000	un coin perdu	12

Pour mettre à jour un enregistrement dans cette table, nous faisons :

```
UPDATE élèves SET (nom_élève='Brindzyngue')
WHERE id_élève='1';
```

Après éxécution de la requête, la table est équivalente à :

id_élève	nom_élève	prénom_élève	adresse_éléve	cp_élève	ville_élève	# id_classe
1	Brindzyngue	toto	trifoullie les oyes	66000	un coin perdu	12

Attention, l'absence de condition peut entrainer certains problèmes (modification des champs de tous les enregistrements par exemple) qui ne provoque pas d'erreur SQL mais qui falsifie toutes les données de la base (les anciennes "bonnes" données risque fort d'être définitivement perdues !!!).

Instruction DELETE

Cette instruction sert à effacer un enregistrement d'une table de la base de données. La syntaxe est la suivante :

```
DELETE FROM <nom_de_la_table>
[ WHERE <condition> ] ;
```

Reprenons notre exemple : **élèves**(**id_élève**, nom_élève, prénom_élève, adresse_éléve, cp_élève, ville_élève, **# id_classe**)

Actuellement la table ressemble à cela :

id_élève	nom_élève	prénom_élève	adresse_éléve	cp_élève	ville_élève	# id_classe
1	brindzingue	toto	trifoullie les oyes	66000	un coin perdu	12

Nous allons supprimer l'enregistrement en faisant :

```
DELETE FROM élèves
WHERE id_élève='1' ;
```

La table est désormais vide.

Attention à la condition. Si elle n'est pas présente, la requête s'applique à tous les enregistrements de la table. En d'autres termes, vous videz la table.
