# LMD SQL PostgreSQL

```
SELECT [ALL | DISTINCT] < projection>
FROM <objets>
[WHERE <clause de restriction> ]
[GROUP BY <clause de groupage> [HAVING <condition>]]
[ORDER BY <clause d'ordonnancement>] ;
```

**PRODUIT CARTESIEN :** utilisation dans la clause FROM

- <nom_table1> [alias] CROSS JOIN <nom_table2> [alias]

**JOINTURE:** utilisation dans la clause FROM de l'opérateur JOIN avec l'une des syntaxes suivantes :

- <nom_table1> [alias] INNER JOIN <nom_table2> [alias] ON <condition> ou
- <nom_table1> [alias] {LEFT | RIGHT | FULL} OUTER JOIN <nom_table2> [alias] ON <condition>
- <nom_table1> [alias] NATURAL JOIN <nom_table2> [alias]

**OPERATEURS ENSEMBLISTES** : UNION | INTERSECT | EXCEPT entre deux requêtes SELECT

**INSERT** :
INSERT INTO <nom_table> [(<liste de nom_colonne>)] VALUES(<liste de valeurs>) ;
INSERT INTO <nom_table> [(<liste de nom_colonne>)] requête ;

**DELETE** : destruction de n-uplets dans **une** table
DELETE [FROM] <nom_table> : efface tous les n-uplets de la table.
DELETE [FROM] <nom_table> WHERE <condition> : efface tous les n-uplets sélectionnés par la condition.

**UPDATE** : permet de modifier la valeur de colonnes de n-uplets d'**une** table.
UPDATE <table> SET nom_colonne = <expression> | ( requête) [WHERE <condition>] ;

**WITH** [ RECURSIVE ] *requête_with* [, ...] SELECT ...
La clause WITH vous permet de spécifier une ou plusieurs sous-requêtes qui peuvent être utilisées par leur nom dans la requête principale. Les sous-requêtes se comportent comme des tables temporaires ou des vues pendant la durée d'exécution de la requête principale. Toutes les requêtes dans la liste WITH sont évaluées. Elles jouent le rôle de tables temporaires qui peuvent être référencées dans la liste FROM de la requête principale.

# LDD LDC SQL PostgreSQL

**Création d'un schéma de table**
CREATE TABLE <nom_table> ( <déclaration colonne>[ , {<déclaration colonne> | <contrainte table>}]$^n$) ;

<déclaration colonne> : <nom-colonne> <type> [<contrainte colonne> [<contrainte colonne>]$^n$]
Principaux types de données
<type> : CHAR(n) | DATE | NUMERIC[(n[,p])] | VARCHAR(n)
<contrainte colonne> : CONSTRAINT <nom_contrainte>
{PRIMARY KEY | [NOT] NULL |
UNIQUE | REFERENCES <nom_table> [(<nom_colonne>)] |
CHECK (<condition>)}
<contrainte table> : CONSTRAINT <nom_contrainte>
{ PRIMARY KEY(<nom_colonne>[,<nom_colonne>]$^n$)
| [NOT] NULL |
UNIQUE (<nom_colonne>[,<nom_colonne>]$^n$) |
FOREIGN KEY (<nom_colonne>[,<nom_colonne>]$^n$)
REFERENCES <nom_table> [(<nom_colonne>[,<nom_colonne>]$^n$)] |
CHECK (<condition>) }

**Création avec peuplement**: CREATE est suivi d'une requête dont le résultat est chargé dans la table déclarée CREATE TABLE <nom_table> (…) AS (requête).

**Destruction d'un schéma** : DROP TABLE <nom_table> (détruit le contenu éventuel **et** le schéma de la table).

**Modification d'un schéma** :
ALTER TABLE <nom_table> ADD (déclaration de colonne ou de contrainte) ;
ALTER TABLE <nom_table> DROP (colonne ou contrainte)
ALTER TABLE <nom_table> ALTER COLUMN (nom_de colonne , avec nouveau type éventuel et/ou une nouvelle Contrainte).

**Fonctions d'agrégat :**

Fonction COUNT() :

La fonction COUNT() est une fonction d'agrégat qui permet d'obtenir le **nombre de n-uplets** qui correspond à une requête donnée.

SELECT COUNT(*) FROM <nom_table> WHERE condition ; renvoie le **nombre de n-uplets, y compris les valeurs en doublons**.
*Exemple* : SELECT COUNT(*) FROM employes ; vous donnera le nombre d'employés.

SELECT COUNT(<attribut1>,<attribut2>, ...) FROM <nom_table> WHERE condition ; renvoie le **nombre de n-uplets correspondant aux valeurs d'attributs (**ici <attribut1>,<attribut2>, …) donnés dans la clause de projection du SELECT, y compris les doublons.
*Exemple* : SELECT COUNT(course_id, student_id) FROM enrolment ; vous donnera le nombre de couples de valeurs de course_id, student_id dans la table enrolment ;

SELECT COUNT(DISTINCT <attribut1>,<attribut2>, ...) FROM <nom_table> WHERE condition ; renvoie le **nombre de n-uplets correspondant aux valeurs d'attributs (**ici <attribut1>,<attribut2>, …) donnés dans la clause SELECT, sans doublon.
*Exemple* : SELECT(DISTINCT nom_emp, prenom_emp) FROM employes ; vous donnera le nombre de valeurs distinctes de couples (nom_emp, prenom_emp), donc sans homonyme.

Fonction SUM() :

La fonction SUM() est une fonction d'agrégat qui renvoie la somme de valeurs correspondant à l'attribut donné en argument sur le résultat de la requête (ou de valeurs distinctes si on précise DISTINCT avant l'attribut ou l'expression). L'attribut doit être un nombre, c'est-à-dire d'un type dérivé du type NUMERIC, comme INTEGER, FLOAT, etc.
*Exemple* : SELECT SUM(salaire) FROM employes ; vous donnera la somme des salaires, autrement dit, la masse salariale de votre entreprise.

Fonction AVG()

La fonction AVG() est une fonction d'agrégat qui renvoie la moyenne non-pondérée des valeurs correspondant à l'attribut donné en argument sur le résultat de la requête.
*Exemple* : SELECT AVG(salaire) FROM employes ; vous donnera la moyenne des salaires de votre entreprise.

Fonction MIN() et MAX()

Ces fonctions d'agrégat vous renvoient respectivement les minimum et maximum sur l'ensemble des valeurs d'attributs correspondant au résultat de la requête. Ces fonctions conviennent pour des nombres, des chaines de caractères (ordre lexicographique en général, alphabétique sur des lettres), sur des dates (respectivement la plus ancienne et la plus récente des dates).
*Exemples* :
-- *Le salaire le plus haut*
SELECT MAX(salaire) FROM employes ;
-- *Le nom de l'employé le premier dans l'ordre alphabétique*
SELECT MIN(nom_emp) FROM employes ;
-- *les dernières commandes en date*
SELECT * FROM commande WHERE date_com >= SELECT MAX(date_com) FROM commande ;