**Documentation sur MCD, MLD et SQL**

# **Notion Analyse et Conception**

1. **Merise**

MERISE est un acronyme signifiant Méthode d’Étude et de Réalisation Informatique par les  
Sous-ensembles ou pour les Systèmes d’Entreprise.  
MERISE est née en 1977 sous l’égide du Ministère Français de l’Industrie qui, dans le souci  
d’apporter une solution à l’organisation, a financé le projet de conception d’intérêt national  
avec des SSII, le Ministère de l’Equipement et des Universitaires.  
MERISE est libre de droit, c’est un Open Source. Elle vise les SI construits autour des Bases  
de données Relationnelles.

MERISE est encore aujourd’hui la méthode la plus connue et la plus utilisée en France, même  
si elle est fortement concurrencée par les méthodes à Objets (UML). Il en existe des multiples  
versions (Merise, Merise 2, Merise Objet, …)  
ANALYSE/CONCEPTION  
**DOMAINE ET PROCESSUS**Le **domaine** correspond à une division de l’organisation en grandes fonctions  
Exemple: Gestion Commerciale, Gestion du Personnel  
Le **Processus** (taches enchainées d’un domaine) est lié à un même évènement déclencheur, en  
général externe et concernant un ensemble de données.  
Exemple: Gestion commerciale d’une entreprise BTP  
-Demande de devis –––> Calcul de devis  
-Fin des travaux –––> Facturation  
-Paiement –––> Gestion règlement  
**a) Le Schéma directeur**Il est mené par la Direction et les Responsables informatiques par domaine. Il englobe l’étude  
globale du SI (choix des grandes orientations, d’organisation et d’informatisation, choix  
techniques généraux: Réseau client/server), un plan de développement de l’informatique de 3 à  
5 ans, une définition des moyens en personnels, matériels et logiciels.  
**b) L’Etude préalable**Elle est menée par Projet et comporte:  
✓ Une analyse critique du système existant (physique, organisationnel, conceptuel)  
✓ Les objectifs du nouveau système (conceptuel, organisationnel)  
✓ Les différents scénarios de solution informatiques  
✓ Une évaluation des coûts et moyens nécessaires  
✓ Un planning de réalisation  
Elle se traduit par un dossier d’étude préalable ou dossier de choix et aboutit au choix d’une  
solution par la Direction.  
c) L’Etude détaillée  
Elle est menée par projet et permet de :  
✓ Préciser l’organisation détaillée de la solution retenue  
✓ Définir logiquement les données et les traitements informatiques de la solution  
✓ Construire le planning de réalisation  
Elle se traduit par un cahier de charge de l’application (contrat vis-à-vis des utilisateurs), un  
dossier d’étude détaillée pour les analystes programmeurs et un cahier de charge  
Matériel/Logiciel pour appel d’offre.  
**d) L’Etude technique**Elle est menée par application et effectue:

• La spécification technique (niveau physique): Structure physique des données,  
décomposition de l’application en programme, dessin d’écran et d’état de sortie  
• La production de programme: elle fournit une documentation technique (maintenance  
des programmes), une documentation utilisateur (Manuel d’utilisation de l’application)  
et un manuel d’exploitation (pour le service exploitation sur gros site informatique)

1. **UML**

UML, c’est l’acronyme anglais pour « Unified Modeling Language ». On le traduit par « Langage de modélisation unifié ». La notation UML est un **langage visuel** constitué d’un ensemble de schémas, appelés des **diagrammes**, qui donnent chacun une vision différente du projet à traiter. UML nous fournit donc des diagrammes pour **représenter** le logiciel à développer : son fonctionnement, sa mise en route, les actions susceptibles d’être effectuées par le logiciel, etc.

**Analyse  
Analyse du domaine**: identifier les éléments du  
domaine ainsi que les relations et interactions  
entre ces éléments.  
**Analyse de l’existant:** déterminer les fonctions  
principales du système.  
**Analyse organisationnelle:** déterminer la  
structure de l’organisation.  
**Analyse technique:** recenser les équipements  
informatiques en place.

**Conception**Définir l’architecture du logiciel  
Définir chaque constituant du logiciel  
– Informations traitées  
– Opérations effectuées  
– Résultats fournis  
– Contraintes à respecter  
Optimiser les modèles  
Choisir un langage de programmation

1. **Etude comparative**

L'approche Merise propose une approche descendante où le système réel est décomposé en activités, elles-mêmes déclinées en fonctions. Les fonctions sont composées de règles de gestion, elles-mêmes regroupées en opérations. Ces règles de gestion au niveau conceptuel génèrent des modules décomposés en modules plus simples et ainsi de suite jusqu'à obtenir des modules élémentaires... Les limites d'une telle approche résident dans le fait que les modules sont difficilement extensibles et exploitables pour de nouveaux systèmes.

L'approche UML : Les fonctions cèdent la place aux cas d'utilisation qui permettent de situer les besoins de l'utilisateur dans le contexte réel. A chaque scénario correspond des diagrammes d'interaction entre les objets du système et non pas un diagramme de fonction...

**Dualité des données -traitements**

L'approche Merise propose de considérer le système réel selon deux points de vue : un point de vue statique (les données), un point de vue dynamique (les traitements). Il s'agit d'avoir une vision duale du système réel pour bénéficier de l'impression de relief qui en résulte, et donc consolider et valider le système final.

L'approche UML : L'approche objet associe les informations et les traitements. De cette façon, elle assure un certain niveau de cohérence.

# **II)​ ​Concepts Analyse et Conception**

# **Modèle Conceptuel des Données (MCD)**

Le Modèle Conceptuel des Données introduit la notion d’entités, de relations et de propriétés.  
Nous allons commencer par voir certains aspects « théoriques » avant de plonger dans la  
pratique. Il décrit de façon formelle les données utilisées par le système d’information. La  
représentation graphique, simple et accessible, permet à un non informaticien de participer à  
son élaboration. Les éléments de base constituant un modèle conceptuel des données sont :  
• les propriétés (ou Attributs) ;  
• les entités ;  
• les relations.  
• Les cardinalités

1. **Les propriétés (ou Attributs)**

Les propriétés sont les informations de base du système d’information.  
Un client possède un numéro de client, un nom, un prénom, habite à une adresse précise, etc.  
Ces informations élémentaires essentielles sont des **propriétés**.  
Les propriétés disposent d’un type. Elles peuvent être numériques, représenter une date, leur  
longueur peut être aussi définie. Par exemple : le nom est une propriété de type alphabétique et  
de longueur 50, c’est-à-dire que la valeur saisie ne comportera aucun chiffre et ne dépassera  
pas cinquante caractères.  
Les types ne sont pas décrits au niveau conceptuel, car ce niveau est trop proche de la définition  
du système physique. Nous y reviendrons plus tard.

1. **Les entités ou objets**

Comme il est aisé de le constater, les clients sont définis par certaines propriétés (numéro, nom,  
prénom…). Le fait de les regrouper amène naturellement à créer une entité Clients.

1. **Les cardinalités**

Elles expriment le nombre de fois où l’occurrence d’une entité participe aux occurrences de la  
relation.  
Dans notre exemple on peut se poser les questions suivantes :  
• Combien de fois au minimum un client peut-il commander un article ?  
• Combien de fois au maximum un client peut-il commander un article ?  
À la première question, nous pouvons répondre qu’un client, pour être client, doit commander  
au moins un article.  
À la deuxième question, nous pouvons répondre qu’un client peut commander plusieurs articles.  
Voici comment symboliser cet état :

|  |
| --- |
| Clients |
| **Numéro Client**  Nom  Prénom  Adresse  Code Postal  Ville |

Cardinalité maximale

|  |
| --- |
| Articles |
| **Numéro Articles**  Désignation  Prix d’achat  Prix de vente |

1 , n

Commander

Cardinalité minimale

Le n représente la notion de « plusieurs » ; ici nous avons représenté le fait qu’un client peut  
commander un ou plusieurs articles. Il faut que nous nous posions les mêmes questions pour  
l’article :  
• Combien de fois au minimum un article peut-il être commandé par un client ?  
• Combien de fois au maximum un article peut-il être commandé par un client ?  
Pour le minimum, nous pouvons l’interpréter de la façon suivante :  
A-t-on des articles qui ne peuvent jamais être commandés ?  
Si nous répondons par « oui », dans ce cas la cardinalité minimale est 0.  
Pour le maximum :  
A-t-on des articles qui peuvent être commandés plusieurs fois ?  
Nous pouvons espérer que oui, dans ce cas la cardinalité maximale sera n.

1. **Formalisme de Représentation**

Le formalisme de représentation d’une dépendance fonctionnelle est le suivant :  
Numéro adhérent → (Nom adhérent, prénom, adresse, code postal, ville, téléphone, mail, date  
d’adhésion)

1. **Dépendances fonctionnelles composées**

Une dépendance fonctionnelle qui comporte plusieurs attributs est dite composée.

*Voici un exemple de dépendance fonctionnelle composée :  
(NOBON, REFPROD) → (QTE)*Connaissant le numéro du Bon de commande et la Référence du produit, nous connaissons de  
façon certaine la Quantité d’un produit dans une commande précise.  
**2. Dépendance fonctionnelle élémentaire**Une dépendance fonctionnelle A → B est élémentaire s’il n’existe pas une donnée C, sousensemble de A, décrivant une dépendance fonctionnelle de type C → B  
*Par exemple :  
REFPROD → DESIGNATION  
NOBON, REFPROD → QUANTITE***3. Dépendance fonctionnelle élémentaire directe**On dit que la dépendance fonctionnelle A → B est directe s’il n’existe aucun attribut C tel que  
l’on puisse avoir A → C et C → B. En d’autres termes, cela signifie que la dépendance  
fonctionnelle entre A et B ne peut pas être obtenue par transitivité.  
*Exemple :  
NumClasse → NumElève  
NumEleve → NomElève  
NumClasse → NomElève*La troisième dépendance fonctionnelle n’est pas directe car nous pourrions écrire :  
NumClasse → NumElève → NomElève  
**4. Méthodologie d’élaboration des dépendances fonctionnelles**L’élaboration des dépendances fonctionnelles est réalisée à l’aide du dictionnaire des données.  
La démarche consiste à rechercher :  
• les dépendances fonctionnelles formées par deux rubriques, élémentaires et directes ;  
• les dépendances fonctionnelles composées.  
**5. Graphe des dépendances fonctionnelles**Le graphe des dépendances fonctionnelles est une étape intéressante car il épure le dictionnaire  
en ne retenant que les données non déduites et élémentaires et il permet une représentation  
spatiale de ce que sera le futur modèle conceptuel des données.  
Voici un graphe des dépendances fonctionnelles :

**NOBON**  REFPROD

DATEBON  
 QTE DESIGN PU

**Modèle Logique des Données (MLD)**  
Le **M**odèle **L**ogique des **D**onnées (**MLD**) est une déduction du MCD. Son but est de nous  
rapprocher au plus près du modèle physique. Pour cela, nous partons du Modèle Conceptuel  
des Données et nous lui enlevons les relations, mais pas n’importe comment, il faut en effet  
respecter certaines règles.  
• Toute Entité du MCD devient une **Table** du MLD qui reprend au minimum les attributs  
de l’entité comme champs de données.  
• Une Association en fonction de ses cardinalités se traduit éventuellement sous forme de  
**table**.  
• L’Identifiant ou clé de chaque entité devient une **clé primaire** de la table.  
• Une **clé étrangère** (Clé primaire d’une autre table) peut être ajoutée comme simple  
champ à la table d’une entité de manière à représenter une association hiérarchique avec  
une autre entité. La table du côté où la cardinalité vaut (**X, 1**) reçoit la clé étrangère.

**Cas (0, n) – (1,1) ou (1,n) – (0,1)**

Voici un modèle conceptuel de départ :

|  |
| --- |
| **COMMANDE** |
| NOBON  DATEBON |

|  |
| --- |
| **CLIENT** |
| COCLI  NOMCLI  RUCLI  VILCLI |

0, N 1,1

Passer

1,1

Obtenir

|  |
| --- |
| **REPRESENTANT** |
| COREP  NOMREP |

0, N

En considérant la relation PASSER, on obtient une migration de la clé primaire de la table  
CLIENT vers COMMANDE comme clé étrangère.  
En considérant la relation OBTENIR, on obtient une migration de la clé primaire de la table  
REPRESENTANT vers COMMANDE comme clé étrangère.

|  |
| --- |
| **CLIENT** |
| COCLI  NOMCLI  RUCLI  VILCLI |

|  |
| --- |
| **COMMANDE** |
| NOBON  DATEBON  #COCLI  #COREP |

|  |
| --- |
| **REPRESENTANT** |
| COREP  NOMREP |

**Cas (0,n) – (0,n) ou (1,n) – (1,n)**Dans le cas où la cardinalité maximale est **n** de chaque côté de la relation, celle-ci se transforme  
en Table ayant comme clé primaire la combinaison des clés primaires des tables en relation et  
les éventuels attributs de l’association.

**CAS PARTICULIERS  
Cas d’une Association binaire de type (0,1)-(1,1)**On ajoute à la table de l’entité qui se situe du côté où la cardinalité vaut (1,1) les attributs  
suivants : une clé étrangère (identifiant de l’autre entité) et les éventuels attributs de  
l’association.

**Cas d’une Association binaire de type (0,1)-(0,1)**On ajoute à l’une ou l’autre table les attributs suivants : une clé étrangère (identifiant de l’autre  
entité) et les éventuels attributs de l’association.

# **Règles de passage du MCD au MLDR**

**Règle 1**

Toute entité devient une table dans laquelle les attributs  
deviennent les colonnes.  
L’identifiant de l’entité constitue alors la clé primaire de la table

**Règle 2**

Une association binaire de type 1 : n disparaît, au profit d’une  
clé étrangère dans la table coté 0,1 ou 1,1 qui référence la clé  
primaire de l’autre table. Cette clé étrangère ne peut pas  
recevoir la valeur vide si la cardinalité est 1,1

**Règle 3**

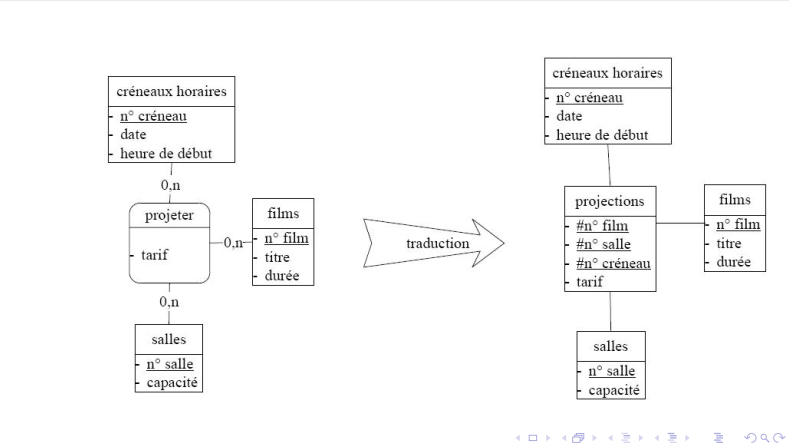
Une association binaire de type n :m devient une table supplémentaire (table  
de jonction) dont la clé primaire est composée des deux clés étrangères.

**Règle 4**

Une association binaire de type 1 :1 est traduite comme une  
association binaire de type 1 : n sauf que la clé étrangère se  
voit imposer une contrainte d’unicité en plus d’une éventuelle  
contrainte de non vacuité (cette contrainte d’unicité impose à la  
colonne correspondante de ne prendre que des valeurs  
distinctes).

**Règle 5**

Une association non binaire est traduite par une table  
supplémentaire dont la clé primaire est composée d’autant de  
clés étrangères que d’entité en association. Les attributs de  
l’association deviennent les colonnes de cette nouvelle table.

****

# **IV​ .SQL**

# LDD : Langage de Description des Données

***Create***

* **Base de données**

**CREATE DATABASE *NomBdd*;**

* **Table**

**CREATE TABLE nomtable**( nomcol1 typecol1 [contraintecol1],  
nomcol2 typecol2 [contraintecol2],  
...,  
contraintetable1,  
contraintetable2...)

*Exemple :*FOURNISSEUR

(F#,FNOM,STATUT,VILLE)

CREATE TABLE FOURNISSEUR  
(F# CHAR(5) NOT NULL UNIQUE,  
FNOM CHAR(20),  
STATUT NUMBER(3) DEFAULT 10,  
VILLE CHAR(15));;

***DROP*Syntaxe :  
DROP DATABASE *NomBdd;***

DROP TABLE nomtable [RESTRICT|CASCADE];

*exemple :* DROP TABLE FOURNISSEUR;  
\* on ne peut pas supprimer une table créée par un autre utilisateur sauf le DBA  
\* DROP nomtable supprime aussi les index sur la table  
• DROP nomtable ne supprime pas les vues associées à cette table mais les  
rend bien sûr indisponibles

***ALTER***

**Syntaxe :**

**ALTER TABLE nomtable modification ;**

Exemple :

ALTER TABLE Customers

ADD Email varchar (255) ;

**RENAME**

RENAME anciennom TO nouveaunom;

*exemple :* RENAME FOURNISSEUR TO F;

**TRUNCATE**

Renvoie un nombre tronqué à 2 décimales :

SELECT TRUNCATE (135.375, 2) ;

# LMD : Langage de Manipulation des Données

**INSERT**

**INSERT INTO nomtable (col1, col2, … coln) VALUES (val1,  
val2, … valn)**

**DELETE**

**DELETE FROM nomtable WHERE prédicat**

**UPDATE**

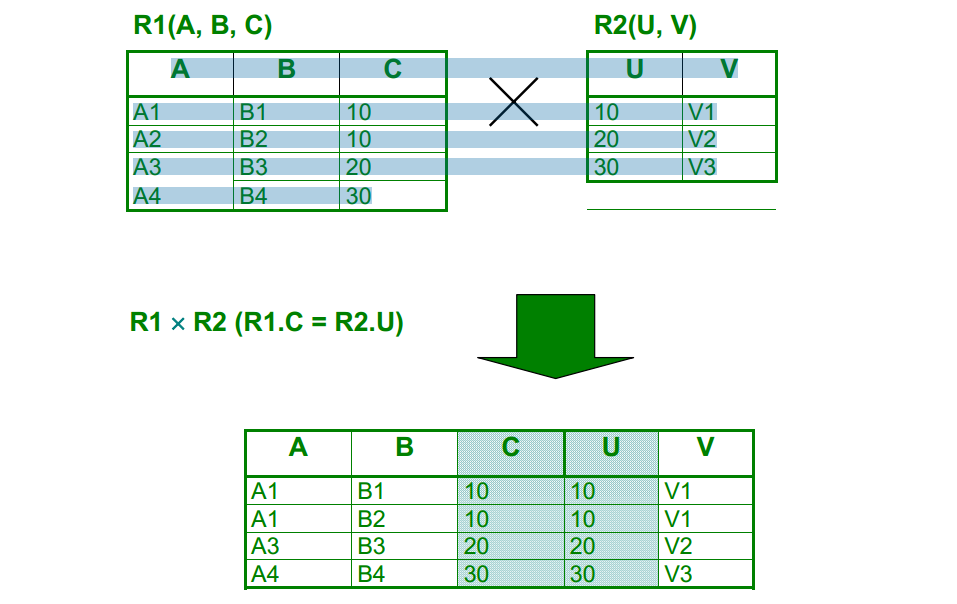
**UPDATE nomtable SET col1 = exp1, col2 = exp2  
WHERE prédicat**

# LID : Langage de d’Interrogation des Données

**PROJECTION**  
La projection d'une relation R1 est la relation R2  
obtenue en supprimant les attributs de R1 non  
mentionnés puis en éliminant éventuellement les nuplets identiques  
On notera :  
**R2 =** π**R1 (Ai, Aj, ... , Am)**la projection d'une relation R1 sur les attributs  
Ai, Aj, … , Am

La projection permet d’éliminer des attributs d’une  
relation  
• Elle correspond à un découpage vertical :

|  |  |  |  |
| --- | --- | --- | --- |
| **A1** | **A2** | **A 3** | **A4** |
|  |  |  |  |
|  |  |  |  |

**JOINTURE**La jointure de deux relations R1 et R2 est une relation  
R3 dont les n-uplets sont obtenus en concaténant les nuplets de R1 avec ceux de R2 et en ne gardant que  
ceux qui vérifient la condition de liaison  
On notera :  
**R3 = R1** × **R2 (condition)**la jointure de R1 avec R2 suivant le critère condition  
• Le schéma de la relation résultat de la jointure est la  
concaténation des schémas des opérandes (s'il y a  
des attributs de même nom, il faut les renommer)  
• Les n-uplets de R1 × R2 (condition) sont tous les  
couples (u1,u2) d'un n-uplet de R1 avec un n-uplet de  
R2 qui satisfont "condition"  
• La jointure de deux relations R1 et R2 est le produit  
cartésien des deux relations suivi d'une restriction  
• La condition de liaison doit être du type :  
<attribut1> :: <attribut2>  
où : attribut1 ∈ 1ère relation et attribut2 ∈ 2ème relation  
:: est un opérateur de comparaison (égalité ou inégalité)  
IUT de Nice - Cours SGBD1 80  
Î La jointure permet de composer 2 relations à l'aide  
d'un critère de liaison

# SQL Sous-requête

Dans le langage SQL une sous-requête (aussi appelé “requête imbriquée” ou “requête en cascade”) consiste à exécuter une requête à l’intérieur d’une autre requête. Une requête imbriquée est souvent utilisée au sein d’une clause WHERE ou de HAVING pou remplacer une ou plusieurs constante.

Syntaxe

Il y a plusieurs façons d’utiliser les sous-requêtes. De cette façon il y a plusieurs syntaxes envisageables pour utiliser des requêtes dans des requêtes.

L’exemple ci-dessous est une exemple typique d’une sous-requête qui retourne un seul résultat à la requête principale.

SELECT \*

FROM `table`

WHERE `nom\_colonne` = (

SELECT `valeur`

FROM `table2`

LIMIT 1

)

Cet exemple montre une requête interne (celle sur “table2”) qui renvoi une seule valeur. La requête externe quant à elle, va chercher les résultat de “table” et filtre les résultats à partir de la valeur retournée par la requête interne.

**A noter :** il est possible d’utiliser n’importe quel opérateur d’égalité tel que =, >,

# ALIAS(AS)

On peut introduire dans la clause FROM un synonyme  
(*alias*) à un nom de table en le plaçant immédiatement  
après le nom de la table  
• Les noms de table ou les synonymes peuvent être  
utilisés pour préfixer les noms de colonnes dans le  
SELECT  
• Les préfixes ne sont obligatoires que dans des cas  
particuliers (par ex. pour une auto-jointure) ; leur emploi  
est cependant conseillé pour la clarté  
• Un alias est utilisé par SQL comme une variable de  
parcours de table (dite *variable de corrélation*) désignant  
à tout instant une ligne de la table

SELECT colonne1 AS c1, colonne2

FROM `table`

# UNION

L'union élimine les doublons, pour obtenir les  
doublons il faut spécifier **ALL** après UNION  
 UNION est une opération binaire, on peut écrire :  
(x UNION y) UNION z ou x UNION (y UNION z)  
 Les parenthèses sont nécessaires dans certains cas,  
par ex. :  
(x UNION ALL y) UNION z  
n'est pas équivalent à  
x UNION ALL (y UNION z)

# GROUP BY

La clause **GROUP BY** permet de partitionner une table  
en plusieurs groupes  
Toutes les lignes d'un même groupe ont la même  
valeur pour la liste des attributs de partitionnement  
spécifiés après GROUP BY  
Les fonctions de calcul opèrent sur chaque groupe de  
valeurs  
 Exemples :  
Q26 Donner pour chaque référence de produit la quantité totale  
vendue  
SELECT V.IdPro, SUM ( V.qte )  
FROM vente V  
GROUP BY V.IdPro  
Q27 Donner la quantité totale achetée par chaque client (0 pour  
ceux qui n'ont rien acheté)

# HAVING

La clause **HAVING** permet de spécifier une condition de  
restriction des groupes  
 Elle sert à éliminer certains groupes, comme WHERE  
sert à éliminer des lignes  
 Exemples  
Q28 Donner les noms des marques dont le prix moyen des  
produits est < 5000F  
SELECT P.marque, AVG ( P.prix )  
FROM produit P  
GROUP BY P.marque  
HAVING AVG ( P.prix ) < 5000  
Q29 Donner les références des produits achetés en qte > 10  
par plus de 50 clients  
SELECT P.marque, AVG ( P.prix )  
FROM vente V  
WHERE V.qte > 10  
GROUP BY V.IdPro  
HAVING COUNT (\*) > 50

# **FONCTIONS**

# COUNT()

permet de compter le nombre d’enregistrement dans une table. Connaître le nombre de lignes dans une table est très pratique dans de nombreux cas, par exemple pour savoir combien d’utilisateurs sont présents dans une table ou pour connaître le nombre de commentaires sur un article.

**Syntaxe**

Pour connaître le nombre de lignes totales dans une table, il suffit d’effectuer la requête SQL suivante :

SELECT COUNT(\*) FROM table

# MAX()

Dans le langage SQL, la fonction d’agrégation MAX() permet de retourner la valeur maximale d’une colonne dans un set d’enregistrement. La fonction peut s’appliquée à des données numériques ou alphanumériques. Il est par exemple possible de rechercher le produit le plus cher dans une table d’une boutique en ligne.

**Syntaxe**

La syntaxe de la requête SQL pour retourner la valeur maximum de la colonne “nom\_colonne” est la suivante:

SELECT MAX(nom\_colonne) FROM table

# MIN()

La fonction d’agrégation MIN() de SQL permet de retourner la plus petite valeur d’une colonne sélectionnée. Cette fonction s’applique aussi bien à des données numériques qu’à des données alphanumériques.

**Syntaxe**

Pour obtenir la plus petite valeur de la colonne “nom\_colonne” il est possible d’utiliser la requête SQL suivante:

SELECT MIN(nom\_colonne) FROM table

# SUM()

Dans le langage SQL, la fonction d’agrégation SUM() permet de calculer la somme totale d’une colonne contenant des valeurs numériques. Cette fonction ne fonction que sur des colonnes de types numériques (INT, FLOAT …) et n’additionne pas les valeurs NULL.

**Syntaxe**

La syntaxe pour utiliser cette fonction SQL peut être similaire à celle-ci:

SELECT SUM(nom\_colonne)

FROM table

# AVG ()

La fonction d’agrégation AVG() dans le langage SQL permet de calculer une valeur moyenne sur un ensemble d’enregistrement de type numérique et non nul.

Syntaxe

La syntaxe pour utiliser cette fonction de statistique est simple :

SELECT AVG(nom\_colonne) FROM nom\_table

**LIMIT()**

La clause LIMIT est à utiliser dans une requête SQL pour spécifier le nombre maximum de résultats que l’ont souhaite obtenir. Cette clause est souvent associé à un OFFSET, c’est-à-dire effectuer un décalage sur le jeu de résultat. Ces 2 clauses permettent par exemple d’effectuer des système de pagination (exemple : récupérer les 10 articles de la page 4).

**ATTENTION :** selon le système de gestion de base de données, la syntaxe ne sera pas pareille. Ce tutoriel va donc présenter la syntaxe pour **MySQL** et pour **PostgreSQL**.

La syntaxe commune aux principaux systèmes de gestion de bases de données est la suivante :

SELECT \*

FROM table

LIMIT 10

LIKE

L’opérateur LIKE est utilisé dans la clause WHERE des requêtes SQL. Ce mot-clé permet d’effectuer une recherche sur un modèle particulier. Il est par exemple possible de rechercher les enregistrements dont la valeur d’une colonne commence par telle ou telle lettre. Les modèles de recherches sont multiples.

Syntaxe

La syntaxe à utiliser pour utiliser l’opérateur LIKE est la suivante :

SELECT \*

FROM table

WHERE colonne LIKE modele

Dans cet exemple le “modèle” n’a pas été défini, mais il ressemble très généralement à l’un des exemples suivants:

* LIKE ‘%a’ : le caractère “%” est un caractère joker qui remplace tous les autres caractères. Ainsi, ce modèle permet de rechercher toutes les chaines de caractère qui se termine par un “a”.
* LIKE ‘a%’ : ce modèle permet de rechercher toutes les lignes de “colonne” qui commence par un “a”.
* LIKE ‘%a%’ : ce modèle est utilisé pour rechercher tous les enregistrement qui utilisent le caractère “a”.
* LIKE ‘pa%on’ : ce modèle permet de rechercher les chaines qui commence par “pa” et qui se terminent par “on”, comme “pantalon” ou “pardon”.
* LIKE ‘a\_c’ : peu utilisé, le caractère “\_” (underscore) peut être remplacé par n’importe quel caractère, mais un seul caractère uniquement (alors que le symbole pourcentage “%” peut être remplacé par un nombre incalculable de caractères . Ainsi, ce modèle permet de retourner les lignes “aac”, “abc” ou même “azc”.

# BETWEEN

L’opérateur BETWEEN est utilisé dans une requête SQL pour sélectionner un intervalle de données dans une requête utilisant WHERE. L’intervalle peut être constitué de chaînes de caractères, de nombres ou de dates. L’exemple le plus concret consiste par exemple à récupérer uniquement les enregistrements entre 2 dates définies.

## Syntaxe

L’utilisation de la commande BETWEEN s’effectue de la manière suivante :

SELECT \*

FROM table

WHERE nom\_colonne BETWEEN 'valeur1' AND 'valeur2'

La requête suivante retournera toutes les lignes dont la valeur de la colonne “nom\_colonne” sera comprise entre **valeur1** et **valeur2**.

# IN

L’opérateur logique IN dans SQL  s’utilise avec la commande WHERE pour vérifier si une colonne est égale à une des valeurs comprise dans set de valeurs déterminés. C’est une méthode simple pour vérifier si une colonne est égale à une valeur OU une autre valeur OU une autre valeur et ainsi de suite, sans avoir à utiliser de multiple fois l’opérateur OR.

Syntaxe

Pour chercher toutes les lignes où la colonne “nom\_colonne” est égale à ‘valeur 1’ OU ‘valeur 2’ ou ‘valeur 3’, il est possible d’utiliser la syntaxe suivante:

SELECT nom\_colonne

FROM table

WHERE nom\_colonne IN ( valeur1, valeur2, valeur3, ... )

**A savoir :** entre les parenthèses il n’y a pas de limite du nombre d’arguments. Il est possible d’ajouter encore d’autres valeurs.

Cette syntaxe peut être associée à l’opérateur NOT pour recherche toutes les lignes qui ne sont pas égales à l’une des valeurs stipulées.