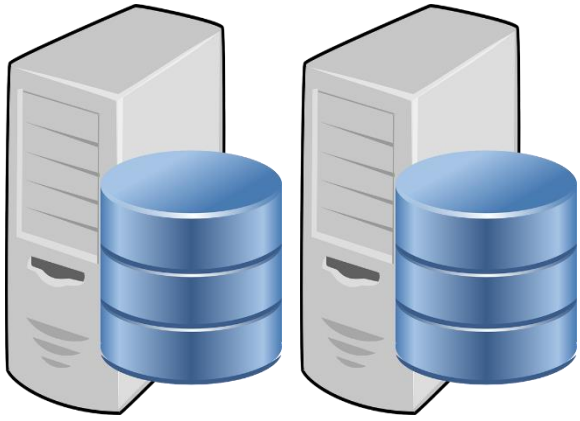
REPUBLIQUE DEMOCRATIQUE DU CONGO

MINISTERE DE L'ENSEIGNEMENT SUPERIEUR ET UNIVERSITAIRE

INSTITUT SUPERIEUR PEDAGOGIQUE DE GEMENA

Département de l'Informatique de Gestion

Cours de Base de données



Ce cours s'adresse aux étudiants de deuxième licence en Informatique de Gestion.

2025

0. INTRODUCTION GENERALE

Vu la multiplicité de données qui circulent autour de nous tant dans un contexte professionnel que social, l'archivage de celles-ci sur des supports électroniques et indélébiles demeure une exigence pour la croissance de nos entités professionnelles ou sociales.

C'est dans cette optique que ce cours de base de données s'adresse aux étudiants de 2è licence en Informatique de Gestion dans le but de les doter des notions théoriques et pratiques liées aux bases de données relationnelles et aux systèmes de gestion de base de données (SGBD en sigle) tels que : MySQL, SQL Server, Access, Oracle et bien d'autres.

Sachant que ce document est un support de cours, il n'a pas la prétention d'être exhaustif ni traiter en détail tous les sujets abordés de façon experte dans le domaine de base de données.

A cet effet, les apprenants sont encouragés à approfondir leurs connaissances dans ce domaine de base de données par des tutoriels, ouvrages et autres canaux à leur portée pour s'affermir davantage dans ce domaine en évolution continue.

Pour les apprenant(e)s, la participation active aux cours et aux travaux pratiques intenses reste un fait conditionnant la réussite.

OBJECTIFS DU COURS

A l'issue de ce cours, l'étudiant(e) sera capable :

- De maitriser les concepts de base utilisés dans le domaine de base de données ;
- De concevoir les bases de données en s'appuyant sur le modèle conceptuel des données (MCD), le modèle logique de données (MLD), le modèle logique des données (MPD) ;
- De maitriser la MERISE ;
- D'utiliser les logiciels de conception de système d'information ;
- De manipuler les données en utilisant le langage SQL ;
- D'implémenter une base de données à partir des SGBD tels que : MySQL, MS-Access.

Chapitre I : Généralités sur les systèmes d'information

I.1. Rappels sur les systèmes d'information

a. Définition

Un système est un ensemble d'éléments en interaction dynamique poursuivant un but commun (selon Joël ROSNAY). Un système est comme quelque chose (n'importe quoi), qui fait quelque chose (activité), qui est dotée d'une structure, pour quelque chose (finalité). Selon Louis LEMOIGNE.

Partant de ces définitions, on peut dire que l'entreprise, l'organisation, l'institution constituent chacune un système.

b. Place du système d'information dans une entreprise

Du point de vue structurel, le système d'information est organisé comme suit :

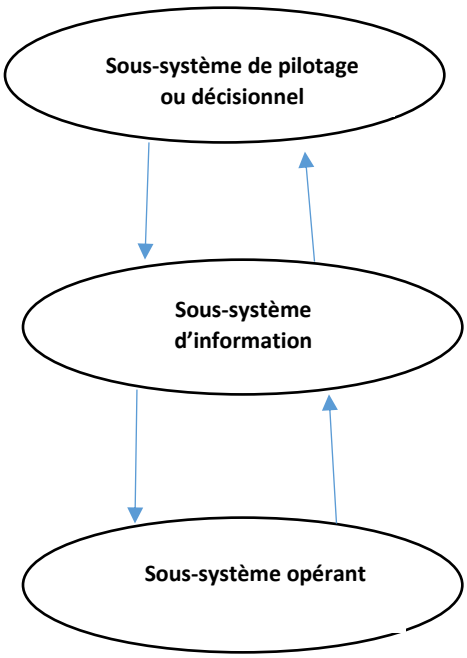


Figure 1 : système d'information dans l'entreprise

- **Le système de pilotage**

Il est composé des dirigeants (les responsables) qui gèrent le système. Il a pour rôle la définition du développement de l'entreprise à travers la détermination des objectifs qui peuvent être à court, moyen ou long terme. Le système de pilotage transmet les ordres au sous-système opérant sous forme informationnelle.

- **Le système opérant**

Il a pour rôle d'exécuter les taches, les opérations selon les ordres provenant du système de pilotage. Il est composé des exécutants.

- **Le système d'information**

Il est l'ensemble des moyens humains, matériels ainsi que des méthodes se rapportant pour traiter les informations au sein de l'entreprise (système). C'est le trait d'union entre le système de pilotage et le système opérant ; son rôle est de faciliter la prise des décisions dans une organisation.

NB : On parle des systèmes informatiques pour désigner toute la structure regroupant les moyens matériels et logiciels mis en place pour pouvoir partager des ressources.

c. Qualités d'un système information

Un bon système d'information doit avoir les qualités suivantes :

- **La rapidité :** le système doit être en mesure de mettre à la disposition des utilisateurs selon leurs besoins, les informations dans un délai très court.
- **La fiabilité :** le système d'information doit être capable de fournir aux utilisateurs les informations utiles comportant moins d'erreurs.
- **La pertinence :** le système d'information doit prendre en charge toutes les informations circulant dans l'entreprise, et trier les données utiles.

- **La sécurité :** le système doit être capable de sécuriser l'accès aux informations.

a. Intégrité
   Les données sont exactes, cohérentes et fiables.
   Les contraintes (clé primaire, clé étrangère, etc.) assurent cette qualité.
b. Sécurité
   Protection contre les accès non autorisés.
   Gestion des droits d'accès selon les utilisateurs (lecture, écriture, modification…).
c. Disponibilité
   Accès facile et rapide aux données à tout moment, pour les utilisateurs autorisés.
d. Redondance minimale
   Les données ne sont pas stockées plusieurs fois inutilement, ce qui évite les incohérences.
e. Partageabilité
   Plusieurs utilisateurs ou applications peuvent accéder à la base en même temps sans conflit.
f. Indépendance des données
   Les données sont indépendantes des applications : on peut modifier la structure sans toucher aux programmes qui les utilisent.
g. Performance
   Temps de réponse rapide pour les opérations de recherche, insertion, mise à jour ou suppression.
h. Fiabilité
   Résistance aux pannes et sauvegarde régulière pour éviter la perte de données.
i. Extensibilité
   Facilité d'ajouter de nouvelles données ou structures sans perturber le système existant.
j. Traçabilité (Auditabilité)

1. Traçabilité

C’est la possibilité de suivre l’historique complet d’une donnée, depuis sa création jusqu’à sa suppression. Cela inclut :

- Les modifications apportées (valeurs avant/après).

- Les utilisateurs ayant effectué ces actions.

- Les dates et heures des actions.

2. Auditabilité

C’est la capacité d’un système à garder des preuves vérifiables des activités qui s’y déroulent. Elle permet :

- De faire des audits de sécurité ou de conformité.

- De détecter les accès non autorisés ou les erreurs humaines.

- De garantir la responsabilité et la transparence dans la gestion des données.

2. Notions sur le système de gestion de fichier

De fortes similitudes existent entre les systèmes de gestion de fichiers utilisés par les différents systèmes d’exploitation et les systèmes de gestion de fichiers de données, du reste objet de cette partie.

a. Fonctionnement d’un SGF

Les fichiers composants le SGF sont des fichiers de données et des fichiers de programmes. Les de programme permettent deux types de traitements :

- Traitement interactif : une action extérieure implique un résultat immédiat de la part du système.
- Traitement par lots : le traitement comporte une série de commandes qui traitent plusieurs éléments et dont le résultat n’est pas immédiat.

D’où, le traitement interactif est un traitement unitaire et le traitement par lots un traitement différé.

b. Objectif d'un SGF

Le système de gestion des fichiers a pour but, en parfaite collaboration avec le système d'exploitation, d'assurer les taches suivantes :

- Gérer automatiquement les mémoires secondaires : disques, bandes, mémoire de masse ;
- Gérer les entrées-sorties ;
- Permettre la création et la suppression des fichiers contenant les données ;
- Permettre les accès en lecture et en écriture ;
- Protéger les fichiers contre les défaillances du système (pannes matérielles ou logicielles) ;
- Permettre le partage des fichiers entre plusieurs utilisateurs ;
- Protéger les fichiers contre les accès non-autorisés ;
- Permettre le langage de programmation pour manipuler et traiter les informations ;

c. Les inconvénients des SGF

Au-delà de l'organisation interne de chaque fichier et des mécanismes permettant de manipuler les enregistrements de ces fichiers, la difficulté majeure demeure dans l'existence de différents fichiers au sein du système d'information d'une entreprise ou d'un organisme. C'est pourquoi, d'autres systèmes de gestion de données se sont mis en place : les systèmes de gestion de base de données (SGBD) ; leur objectif étant d'éliminer les inconvénients indirects.

Chaque génération de SGBD constitue une avancée supplémentaire liée à des innovations technologiques ; chaque génération apporte des solutions nouvelles pour atteindre les objectifs énoncés et pour pallier les contraintes de la génération précédente.

3. Les systèmes de gestion de base de données

a. Définition

Un système de gestion de base de données (SGBD) est un logiciel de haut niveau qui permet de manipuler les informations stockées dans une base de données. La complexité d'un SGBD est essentiellement due à la diversité des techniques mises en œuvre, de la multiplicité des composants intervenant dans son architecture, et de différents types d'utilisateurs (administrateurs, programmeurs, non informaticiens…qui sont confrontés à différents niveaux au système.

En vue de mettre de l'ordre dans tout cela, il importe de se convenir autour d'une architecture standard conforme à la plus grande partie des SGBD existant, et offrant l'avantage de bien illustrer les principales caractéristiques d'un SGBD.

Ainsi, un SGBD présente plusieurs avantages, parmi lesquels nous pouvons citer :

- La souplesse d'accès aux données ;
- La sécurité ;
- Le partage de données ;
- Indépendance de données ;

a. La souplesse d'accès aux données

Comme son nom le désigne, ces systèmes sont destinés pour gérer les données que les utilisateurs auront besoin ultérieurement. Ils doivent permettre d'accéder facilement à n'importe quelle donnée de la base de données. Pour accéder à ces données, le système doit utiliser un langage d'interrogation appelé langage des requêtes (SQL) qui permet d'extraire les données contenues dans la base de données.

b. La sécurité

L'agression extérieure est un facteur perturbateur des données contenues dans la base de données. Les SGBD doivent être capables de protéger les données qu'il gère. Ces agressions peuvent être physiques, comme la panne d'un disque dur, elles peuvent aussi être humaines comme une mauvaise manipulation de la machine par un utilisateur.

c. Partage de données

Jadis, les informaticiens rencontraient d'énormes difficultés pour partager des données avec d'autres utilisateurs d'où la création du domaine de base de données. Différentes opérations doivent s'effectuer sur une même base de données, doivent s'exécuter comme si elles étaient seules à opérer. Sur ce, pour détecter d'éventuels conflits d'accès qui puissent exister entre plusieurs utilisateurs, les SGBD ont facilité le partage de données.

d. Indépendance de données

L'indépendance est l'un des aspects majeurs offerts par un SGBD. Une application manipulant ses données par l'intermédiaire de fichiers est fortement dépendante de ses données. Dans ce cadre, nous distinguons deux niveaux d'indépendance de données :

- L'indépendance physique : permet de modifier la structure de stockage ou les méthodes d'accès aux données sans que cela ait des répercutions au niveau des applications on pourra ainsi ajouter ou supprimer un index sur une collection ou bien changer une méthode de tri.
- L'indépendance logique doit permettre de modifier l'organisations de données sans affecter les utilisateurs. Il permet donc de faire face aux nouveaux besoins, ce qui est indispensable, si l'on considère qu'une base de données est un modèle réel.

e. Le niveau de description de données

Le concept d'indépendance de données est un aspect fondamental d'un SGBD. Pour obtenir cette indépendance, on considère trois niveaux de représentation de base de données : Niveau conceptuel, niveau logique et niveau physique.

Quelques SGBD les plus utilisés dans les entreprises congolaises

1**. MySQL**
- Open source, gratuit, très populaire dans les PME, écoles, projets web.
- Utilisé avec des outils comme PHP, WordPress ou Laravel.

2**. Microsoft SQL Server**
- Très présent dans les banques, entreprises publiques et privées.
- Compatible avec les logiciels de gestion de Microsoft (ex : Dynamics).

3. Oracle Database
- Utilisé dans les grandes entreprises, institutions publiques et opérateurs télécoms.
- Très robuste, mais coûteux.

4. PostgreSQL
- Gratuit, open source, fiable.
- Présent dans les projets universitaires, ONG, star…

[10:38, 07/01/2026] Chat Gptwhatsapp: - Alternative libre à MySQL.
- De plus en plus utilisée pour des raisons de performance et de licence.

9. Firebase
- Utilisée dans les applications mobiles développées localement.

10. Odoo (base PostgreSQL intégrée)
- SGBD indirectement utilisé via la solution de gestion Odoo (ERP), répandue dans plusieurs PME, ONG et institutions.

En résumé, retenons qu'un SGBD est destiné à gérer un gros volume d'informations, persistantes (années) et fiables (protection sur pannes), partageables entre plusieurs utilisateurs et/ou programmes et manipulées indépendamment de leurs représentations physiques.

Que doit- on savoir pour utiliser un SGBD ?

L'utilisation d'un SGBD suppose de comprendre (et donc de savoir utiliser) les fonctionnalités suivantes :

1. Définition du schéma de données en utilisant les modèles de données du SGBD.
2. Opérations sur les données : recherche, mise en jour, modification, ajout etc.
3. Partage de données entre plusieurs utilisateurs. (Mécanisme de transaction).
4. Optimiser les performances, par le réglage de l'organisation physique des données. Cet aspect relève plutôt de l'administration.

Cycle de vie d'une base de données

On appelle cycle de vie d'une base de données, la suite de phase partant de la conception, l'implantation, utilisation. La conception d'une base de données est phase d'analyse qui aboutit à déterminer le futur contenu de la base de données.

Lorsqu'une entreprise décide pour son informatisation d'adopter l'approche de base de données, le premier problème à résoudre peut-être le plus difficile est de déterminer les informations qu'il conviendra de mettre dans la base de données. Il faudra pour cela que l'ensemble des utilisateurs actuels et futurs de cette base de données se mettent d'accord

sur la nature et les caractéristiques des informations qu'il faut garder pour assurer la gestion de l'entreprise.

Une fois que cet accord aura été établi, il faudra pouvoir transmettre son contenu au logiciel SGBD choisi par l'entreprise. Ceci se fera au moyen d'un langage symbolique, spécifique du logiciel choisi, qu'on appelle langage de description de données (LDD). Une fois que le SGBD aura pris connaissance de cette description, il sera possible aux utilisateurs d'entrer les données, c'est-à-dire de constituer la première version, initiale, de la base de données.

On appelle implantation de la base de données, cette phase qui consiste à décrire la base de données dans le langage du SGBD et construire cette première version. Une fois l'implémentation terminée, on peut commencer l'utilisation de la base de données. Celle-ci se fait au moyen d'un langage, dit langage de manipulation de données (MLD), qui permet d'exprimer aussi bien les requêtes d'interrogation (pour obtenir les informations contenues dans la base) que des requêtes de mise à jour, (pour ajouter de nouvelles informations, supprimer les informations périmées, modifier le contenu des informations).

7. Différents modelés des bases de données

a. Modèle hiérarchique

Historiquement le premier, ce modèle est fondé sur la modélisation arborescente de données des pointeurs entre différents nœuds(segments).

Les données sont classées hiérarchiquement selon un graphe arborescent descendant où on a un segment racine unique, des segments internes et des segments feuilles. Le niveau d'un segment caractérise sa distance à la racine.

Exemple des informations sur des vols d'avion

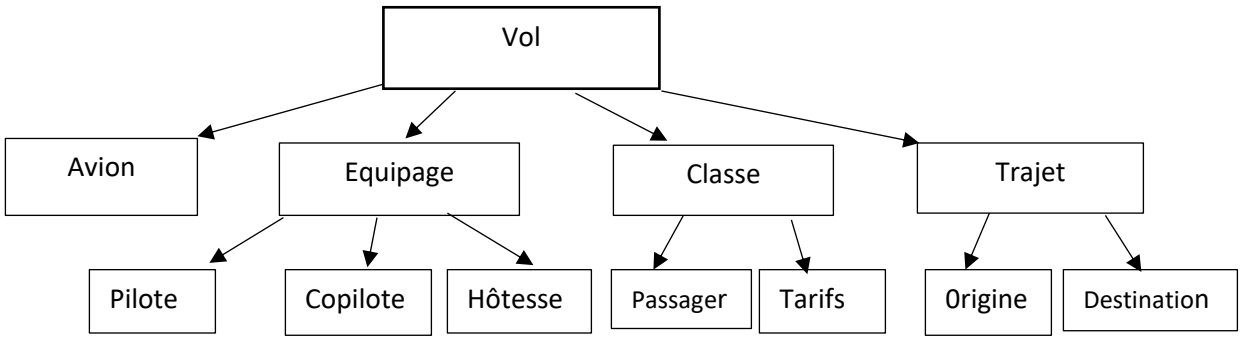


Figure : 2 Modèle hiérarchique

Explications :

- Observant la figure ci-dessus, nous pouvons réaliser que pour trouver un vol où un passager est inscrit il faut parcourir tous les vols pour déterminer si le passager donné est présent.
- La suppression d'un nœud entraine la perte de toutes les informations sur ses descendants.

B. **Modèle réseau**

C'est une extension du modèle hiérarchique, il permet d'établir des connexions entre les différents éléments. Ce modèle utilise de même des pointeurs entre les différents enregistrements, toutefois la structure n'est plus forcément arborescente dans le sens descendant.

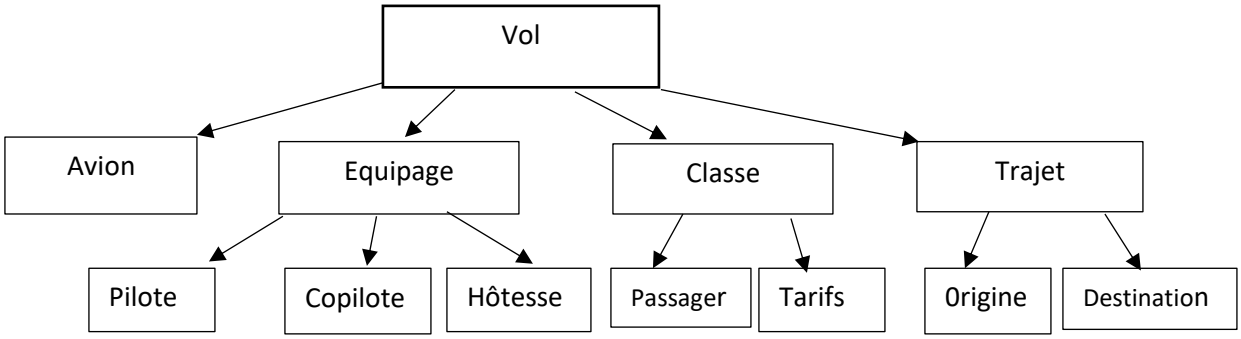


Figure : 3 Modèle réseau

c. Modèle relationnel

Formalisé par Boyce Codd en 1970, ce modèle stocke les données dans les tables sans préjuger de la façon dont les informations sont stockées dans la machine, et l'exploitation est réalisée à l'aide d'un langage déclaratif, à l'instar de : Oracle, MySQL, Access…

Définition :
Une base de données relationnelle organise les données en tables (relations) composées de lignes (enregistrements) et de colonnes (champs). Chaque table a une clé primaire qui identifie de façon unique chaque ligne.

Exemples : MySQL, PostgreSQL, Oracle, SQL Server, SQLite.

Caractéristiques principales :
- Modèle tabulaire : données stockées sous forme de tables.
- Utilise le langage SQL (Structured Query Language) pour manipuler les données.
- Relations entre tables établies via des clés primaires et clés étrangères.
- Données fortement structurées avec un schéma défini à l'avance.
- Intégrité référentielle : contrôle des liens entre les tables.
- Convient aux systèmes transactionnels : banques, administrations, etc.

Une base de données relationnelle est un ensemble des schémas de relation et dont les occurrences sont types e ces relations. Elle est fondée sur théorie mathématique des relations (Union, différence, Produit cartésien…).

Exemple :

T_AGENT				
Matricule	Nom	Prenom	Age	Service

T_SERVICE	
ID_SERVICE	NOM

d. Modèle oriente objet

Ce modèle a été mis au point pour gérer les données multimédias. Il regroupe les concepts essentiels pour modéliser de manière progressive des objets complexes encapsulés par des opérations de manipulations associées. L'organisation de données est faite sous forme d'instances de classes hiérarchisées qui possèdent leurs propres méthodes d'exploitation. Les champs sont les instances de ces classes.

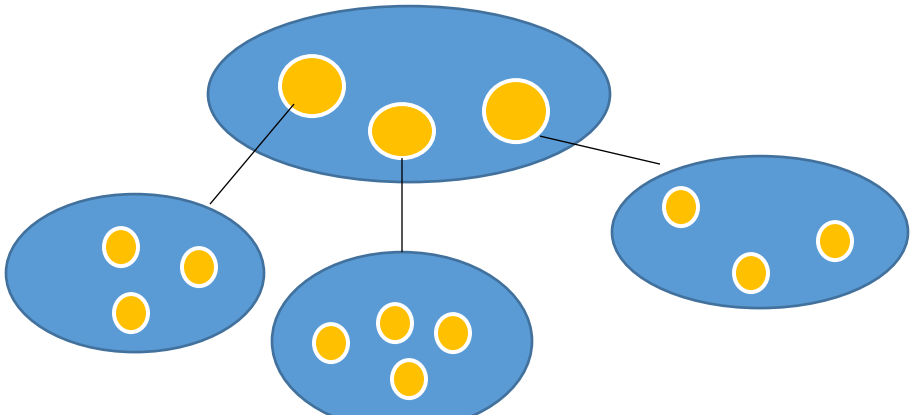


Figure : 4 Modèle objet

e. Modèle déductif

Les données sont représentées sous forme de table, mais leur manipulation se fait par calcule de prédicat.

e. Modèle Non Relationnel

Définition :

Une base de données non relationnelle stocke les données dans des structures variées autres que les tables : documents, paires clé-valeur, graphes, colonnes, etc. Elle est conçue pour la scalabilité et la flexibilité.

Exemples : MongoDB, Cassandra, Redis, CouchDB, Neo4j.

Types et caractéristiques :

a. Document Store (ex : MongoDB)

- Données stockées sous forme de documents JSON ou BSON.
- Schéma flexible (chaque document peut avoir des champs différents).
- Idéal pour les applications web modernes.

b. Key-Value Store (ex : Redis, Amazon DynamoDB)

- Stocke les données sous forme de paires clé-valeur.
- Très rapide, adapté aux caches ou configurations simples.

c. Column Store (ex : Apache Cassandra, HBase)

- Données stockées par colonnes au lieu de lignes.
- Optimisé pour les lectures massives (Big Data).

d. Graph Database (ex : Neo4j)

- Stocke les données sous forme de nœuds et d'arêtes.
- Idéal pour les réseaux sociaux, cartographie, relations complexes.

Chapitre III: Initiation aux logiciels de modélisation JMERISE

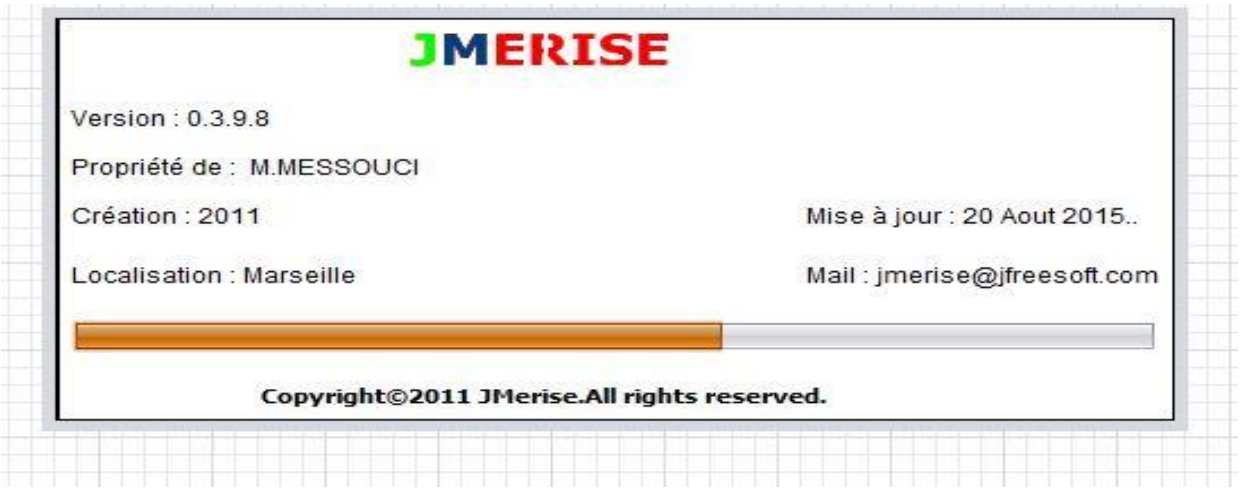
Maintenant que nous avons compris comment concevoir une base de données à partir de l'aperçu d'un objet du monde réel, c'est-à-dire analyser le problème et faire le passage du MCD au MPD en passant par le MLD, il importe de savoir qu'ils existent des logiciels qui permettent de créer le modèle conceptuel des systèmes d'information et de générer automatiquement le modèle physique de données.

Ce chapitre penchera sur l'utilisation des logiciels de conception des systèmes d'information, notamment JMERISE pour nous permettre de gagner du temps.

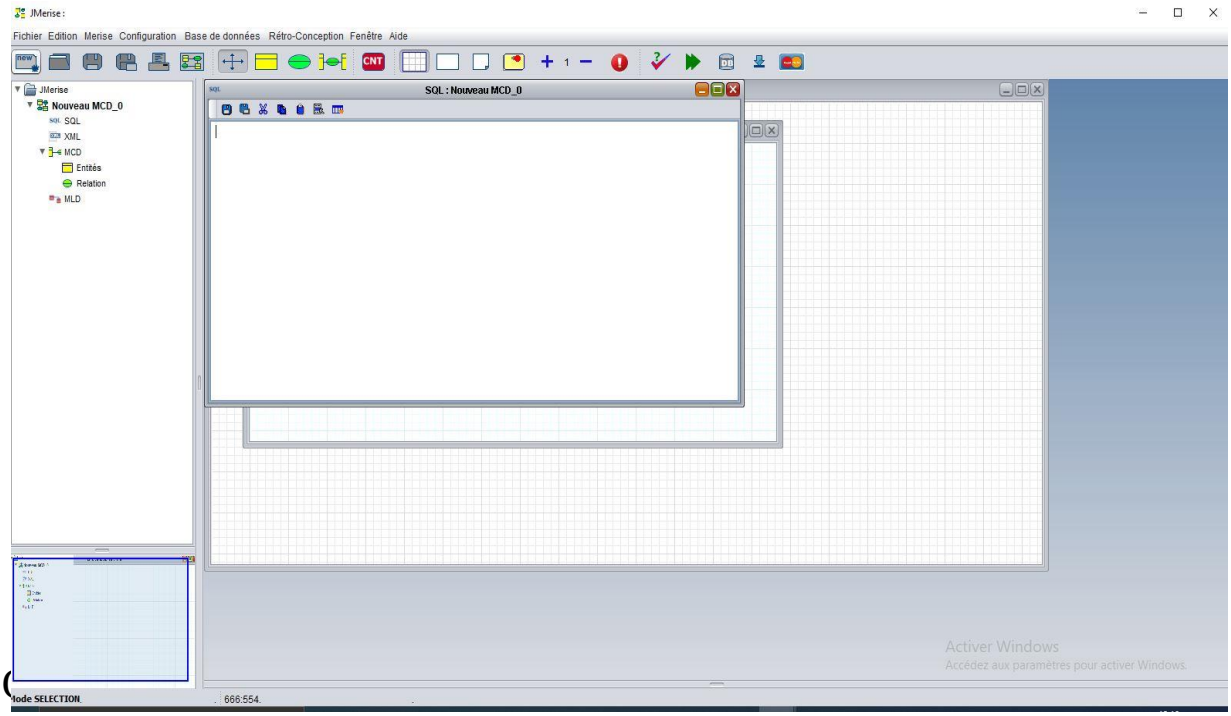
Parmi ces logiciels, nous avons :

- Le DBMAIN
- Le JMerise et autres.

Ces logiciels sont gratuits et très pratiques. Dans le cadre de ce cours, nous apprendrons à modéliser avec le logiciel JMerise. Une fois que le logiciel est télécharger sur le internet, lancez-le pour utilisation et voici en image ce qui sera afficher sur votre écran après avoir double-cliqué sur le fichier **JMerise.jar**.



Après le lancement, l'interface de JMERISE s'affiche et se présente comme suit :

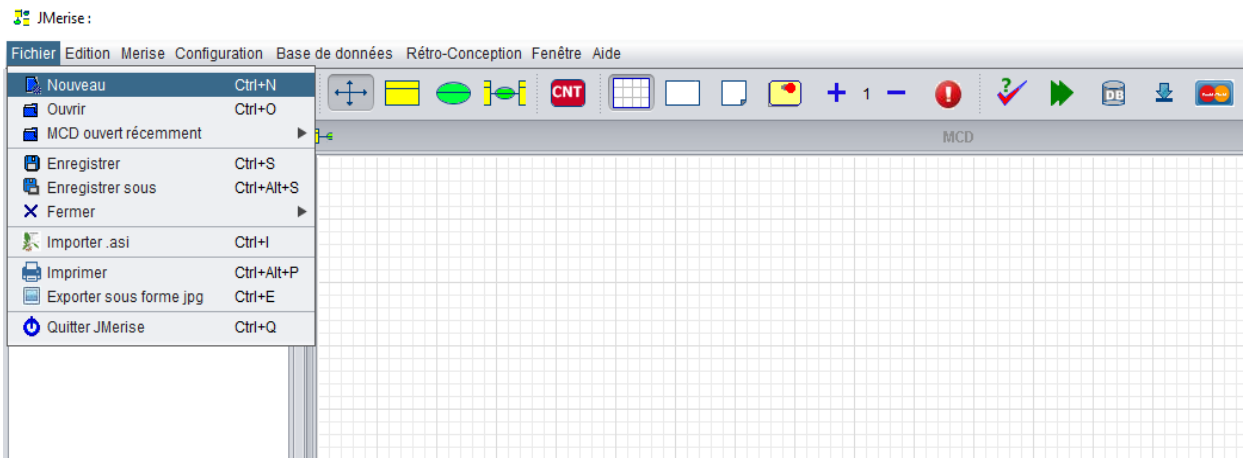


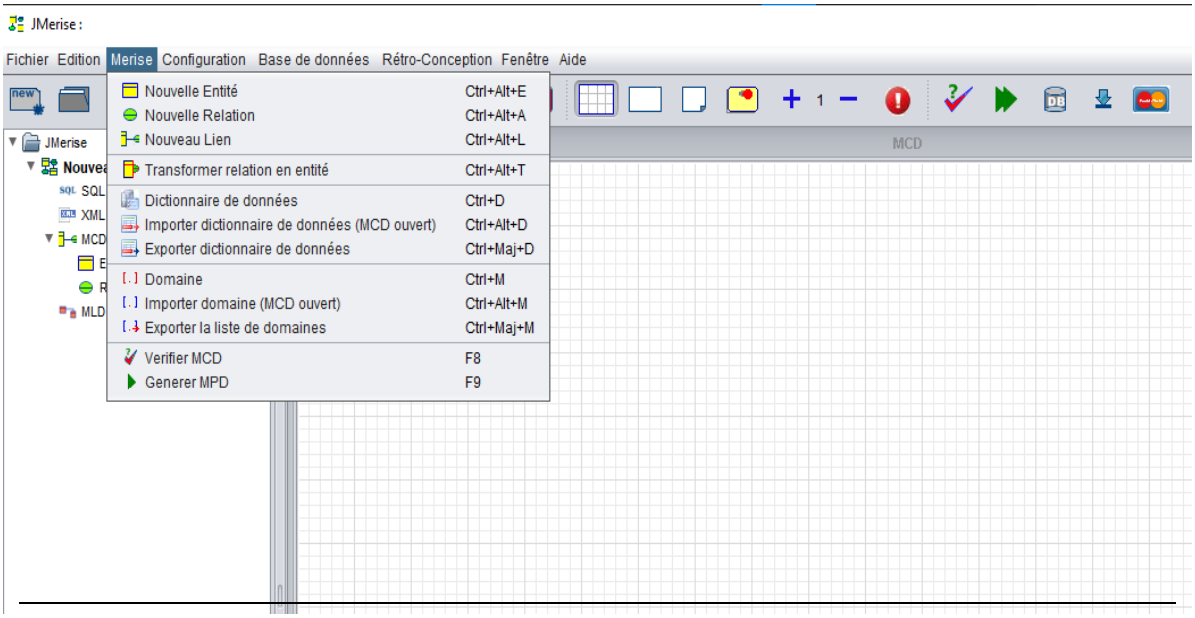
(

menus : **Fichier, Edition, Merise, Configuration…**

Utiliser le menu pour enregistrer le projet, ouvrir un projet existant, nommer et enregistrer un projet dans un emplacement au choix.

Le menu Edition permet de : couper, copier, coller des objets Merise et autres.




JMerise :
Fichier Edition Merise Configuration Base de données Rétro-Conception Fenêtre Aide
Nouvelle Entité Ctrl+Alt+E
Nouvelle Relation Ctrl+Alt+A
Nouveau Lien Ctrl+Alt+L
Transformer relation en entité Ctrl+Alt+T
Dictionnaire de données Ctrl+D
Importer dictionnaire de données (MCD ouvert) Ctrl+Alt+D
Exporter dictionnaire de données Ctrl+Maj+D
Domaine Ctrl+M
Importer domaine (MCD ouvert) Ctrl+Alt+M
Exporter la liste de domaines Ctrl+Maj+M
Verifier MCD F8
Generer MPD F9
MCD

Chapitre IV: Création de base de données avec MySQL

0. Introduction

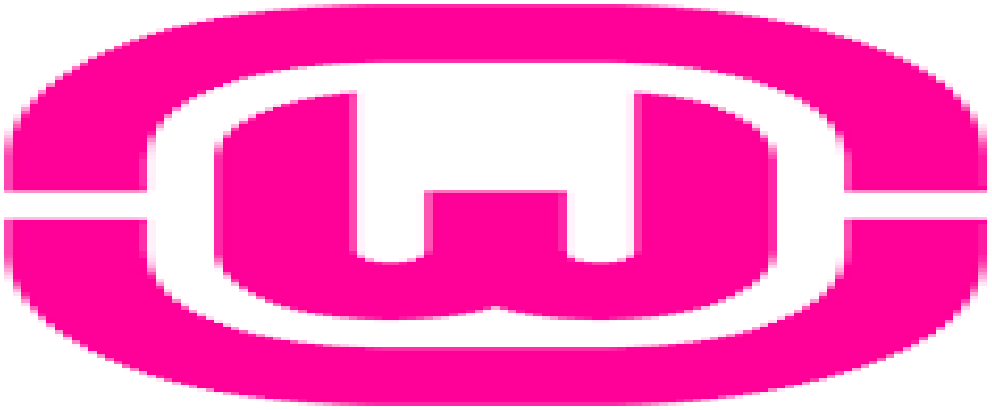
Ce chapitre, nous introduira dans le bain de la création d'une base de données dans un SGBD, en l'occurrence MySQL. Nous allons installer, exploiter en créant des bases de données, des tables, en insérant les données et en les extrayant grâce aux requêtes SQL.

1. Création de base de données

De nos jours, les bases de données servent quasiment dans tous les domaines de la vie courante. Dans le secteur de la santé, dans le secteur de l'éducation, dans l'armée, dans l'administration publique, dans l'enseignement et autre. Bien gérer une base de données, c'est garantir l'intégrité de données c-à-d mettre en place un environnement qui permettra que les données ne soient pas corrompues avant leur utilisation.

Dans ce chapitre, nous apprendrons à créer des bases de données en fonction de nos besoins, pour garantir un maximum de sécurité concernant l'intégrité de ces données à l'intérieur de la base. Et pour cette fin, nous utiliserons **MySQL** comme **SGBD** (Système de Gestion de Base de données).

En image, le logo officiel du WAMP server, servant de logiciel de base pour la création de base de données avec MySQL :



Lors de la pratique, nous l'étudierons en profondeur…

Les termes ci-après constituent les mots de vocabulaires les plus utilisés lors de l'étude de base de données.

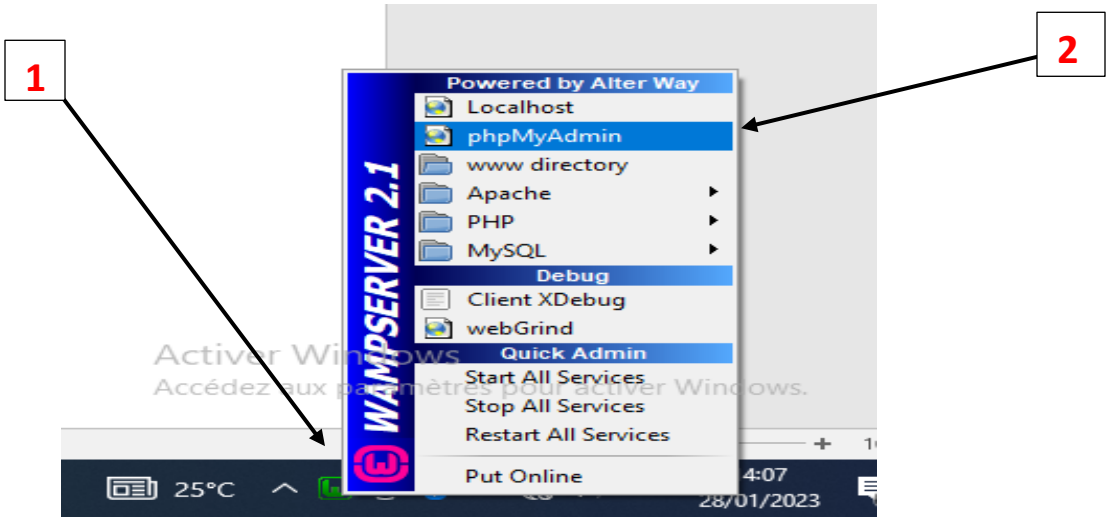
Tables	Une table est l'endroit où les données sont stockées. Une base de données doit au minimum contenir une table. Les informations dans les tables sont stockées dans les lignes et des colonnes.
Colonnes	Les colonnes sont constituées d'une cellule de chaque ligne. Elles donnent une définition à chaque ensemble de définition et ne peut contenir qu'un seul type de données. Chaque table doit contenir au minimum une colonne.
Enregistrements	Les enregistrements sont constituées d'une cellule de chaque colonne. Les lignes sont aussi appelées des records. Une table peut contenir autant de ligne que l'on veut, on est simplement restreint par notre espace disque.
Index	Comme dans un livre permet d'accéder aux données le plus rapidement. Les index correspondent à des listes prédéfinies qui peuvent informer sur la position physique ou non d'une donnée. Ils peuvent être utilisés par SQL pour trouver une ligne de données. Un index ne peut pas couvrir plus d'une table.
Vues	On peut identifier les vues en tant que tables virtuelles

Après que nous nous soyons familiarisés avec les vocabulaires de base de données, il est temps que nous apprenions à utiliser **wamp server** qui est le logiciel faisant office du serveur permettant de stocker les données avec le SGBD MySQL.

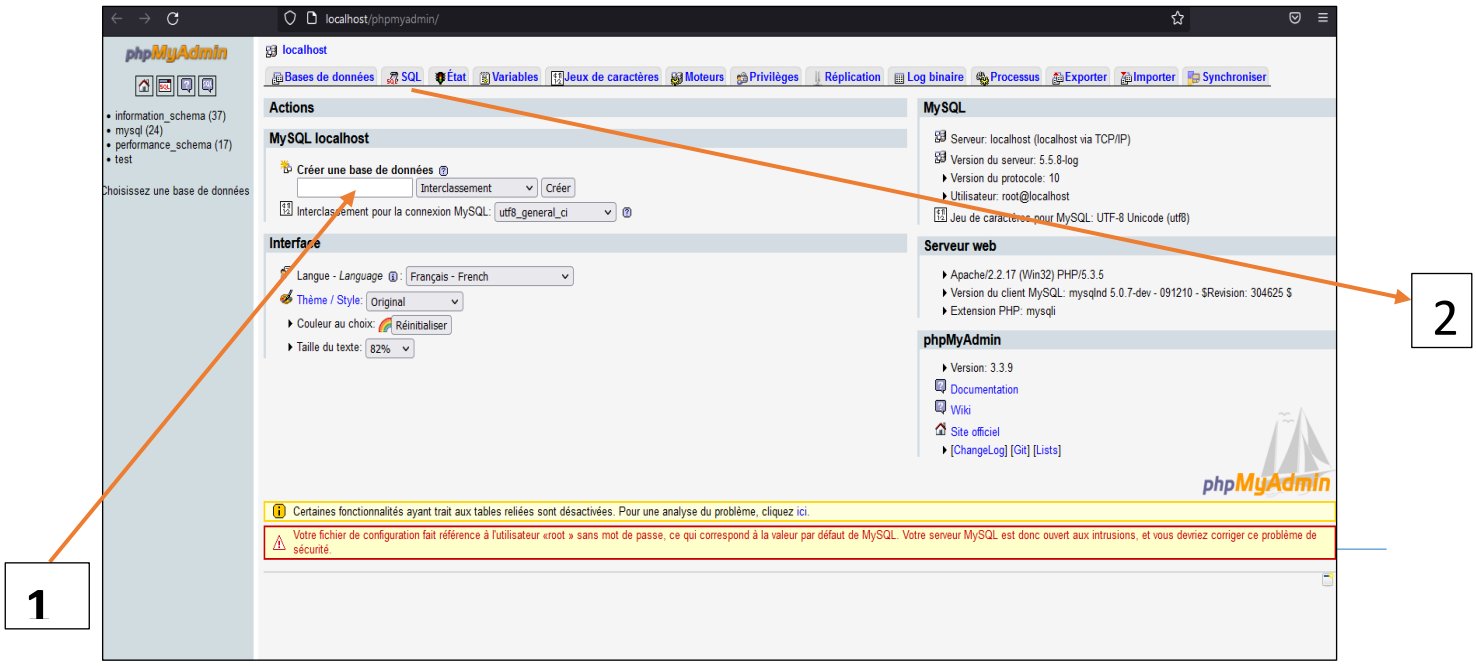
Comment utiliser ce logiciel ?

1. Double-cliquer sur l'icône du logiciel et attendre qu'il s'affiche en couleur bleue.
2. Clic droit sur la petite icone du logiciel en bas de l'écran, côté droit.
3. Choisissez une option sur le menu qui va s'afficher selon votre besoin.
4. Double-cliquer sur l'option choisie

Voir l'image ci-dessous :



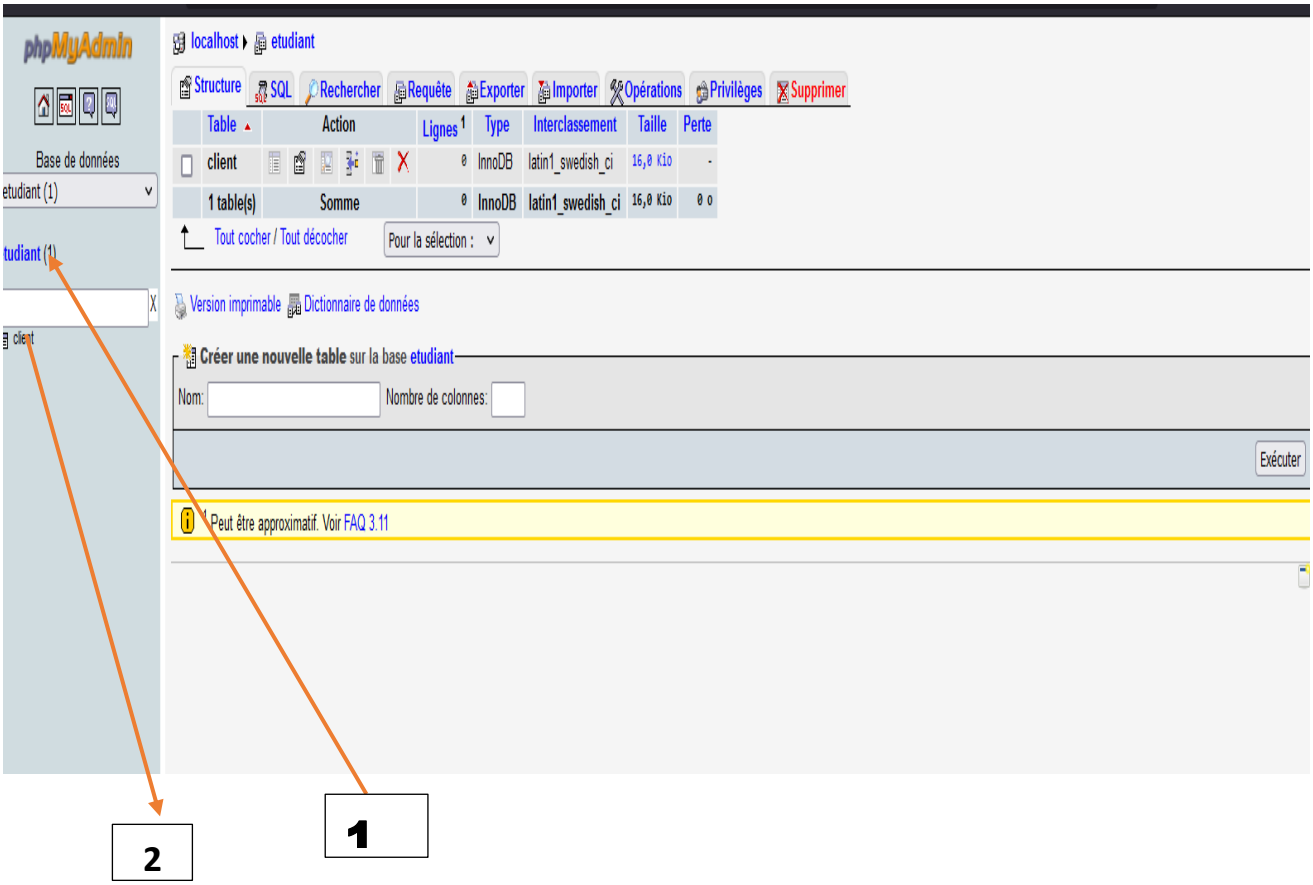
Si vous avez cliqué sur PHP, voici ce que l'interface que vous affichera le logiciel :



1 : Inscrire le nom de la base de données puis appuyer sur exécuter.

2 : Cliquer pour tapez les instructions SQL afin de créer la base de données.

Après avoir fait l'une des opérations ci-dessus, voici ce que sera l'affichage…



- 1 : La base de données étudiant a été créée.
- 2 : La table client a été créée.

I. Les contraintes d'intégrité

Comme vu précédemment, il existe toute une gamme de contraintes pour assurer l'intégrité de données dans la base de données. Les contraintes

s’appliquent exclusivement aux colonnes des tables et possèdent des caractéristiques propres.

a. IDENTITY

Ce type de contrainte peut être affectée à une colonne par table, de type numérique entier. Elle permet d’incrémenter les valeurs d’une colonne ligne après ligne. Par défaut, la contrainte IDENTITY part de1, et a un pas d’incrémentation de1.

b. PRIMARY KEY

Cette contrainte permet de définir une clé primaire sur un ou plusieurs colonnes d’une table. Il ne peut y avoir qu’une seule clé primaire par table, et la ou les colonnes sur lesquelles elle est définie doivent être de type NOT NULL.

c. UNIQUE

La contrainte UNIQUE comme son nom l’indique, va nous permettre de préciser sur une colonne, si les valeurs contenues dans celles-ci ne doivent pas être dupliquées dans plusieurs enregistrements. De ce fait, il ne sera pas possible avec une contrainte UNIQUE d’avoir deux fois une même valeur pour une colonne donnée. Contrairement à une table possédant la clé primaire, une table peut avoir plusieurs colonnes concernées par la contrainte UNIQUE.

d. REFERENCE

La contrainte REFERENCE traduit la liaison qui existe entre la clé primaire et la clé étrangère de deux tables. Il est conseillé de créer ce genre de contrainte qu’après la création de toutes les tables impliquées sinon, lors de la compilation de votre script, des erreurs peuvent apparaitre.

e. CHECK

Cette contrainte permet de vérifier, avant insertion ou mise à jour d’une des données contenues dans la colonne en question que les données à

insérer sont bien au format voulu, ou encore qu'une valeur entrée dans la colonne pour un enregistrement appartiendra à un domaine particulier.

f. DEFAULT

La contrainte DEFAULT est particulièrement utile pour éviter les valeurs NULL dans une table. Il faut garder donc à l'esprit qu'une valeur par défaut ne sera utilisée que dans le cas où l'utilisateur n'entre pas une valeur pour une colonne en particulier.

Chapitre V : le langage SQL (structured query language)

1. Introduction

Dans ce chapitre, il sera question d'étudier en profondeur le langage de manipulation (interrogation) de base de données relationnelle.

2. La commande CREATE DATABASE

La création d'une base de donnée en SQL est possible en ligne de commande. Même si les SGBD (systèmes de gestion de base de données) sont souvent utilisés pour créer une base, il est indispensable de connaitre la commande à utiliser qui est très simple :

CREATE DATABASE ma_base

Avec MySQL, si une base de données porte déjà ce nom, la requête retournera une erreur. Pour éviter d'avoir cette erreur, il est recommandé d'utiliser la requête suivante pour MySQL :

```
CREATE DATABASE IF NOT EXISTS ma_base
```

L'instruction IF NOT EXISTS permet de ne pas retourner une erreur si une base de donnée du même nom existe déjà. La base de données ne sera pas écrasée.

1. LA COMMANDE DROP DATABASE

En SQL, la commande **DROP DATABASE** permet de supprimer totalement une base de données et tout ce qu'elle contient. Cette commande est à utiliser avec beaucoup d'attention car elle permet de supprimer tout ce qui est inclus dans une base de données : les tables, les données, les index…

Syntaxe

Pour supprimer la base de données « ma_base », la requete sera la suivante :

```
DROP TABLE ma_base
```

Attention : n'oublier pas d'effectuer une sauvegarde. Par ailleurs, si le nom de la base de données à supprimer n'existe pas, la requête retournera une erreur. Ainsi, pour éviter cette erreur si vous n'êtes pas sur du nom, utilisez l'option IF EXISTS. La syntaxe sera alors la suivante :

```
DROP DATABASE IF EXISTS ma_base
```

2. LA COMMANDE CREATE TABLE

La commande CREATE TABLE permet de créer une table en SQL. Une table est une entité contenue dans une base de données pour stocker les données ordonnées dans des colonnes. La création d'une table sert à définir les colonnes et le type de données qui seront contenus dans chacune des colonnes (entier, chaine des caractères, date, valeur binaire…)

Syntaxe :

La structure générale pour créer une table est la suivante :

```
CREATE TABLE nom_de_la_table (
Colonne1 type_de_données,
Colonne2 type_de_données,
Colonne2 type_de_données,
) ;
```

Pour chaque colonne, il est possible de définir les éléments tels que :

- NOT NULL : empêche d'enregistrer une valeur nulle dans une colonne.
- DEFAULT : attribue une valeur par défaut si aucune donnée n'est indiquée pour cette colonne lors de l'ajout d'une ligne dans la table.
- PRIMARY KEY : indiquer si cette colonne est considérée comme clé primaire pour un index.

Exemple :

Imaginons que l'on souhaite créer une table utilisateur, dans laquelle chaque ligne correspond à un utilisateur inscrit sur un site web. La requête pour créer cette table peut ressembler à ceci :

```
CREATE TABLE utilisateur (Id INT PRIMARY KEY NOT NULL,
Nom VARCHAR (100),
prenom VARCHAR (100),
email  VARCHAR (100),
date _naissance DATE,
pays VARCHAR (100),
ville  VARCHAR (100),
code_postal VARCHAR (100),
Nbre_tchat INT
)
```

LA COMMANDE ALTER TABLE

La commande ALTER TABLE en SQL permet de modifier une table existante. Il est ainsi possible d'ajouter une colonne, d'en supprimer une ou de modifier une colonne existante, par exemple pour changer le type.

Syntaxe de base

D'une manière générale, la commande ALTER TABLE s'utilise de la manière suivante :

```
ALTER TABLE nom_table
Instruction.
```

Le mot « instruction » ici sert à désigner une commande supplémentaire qui sera détaillée ci-dessous selon l'instruction que l'on souhaite effectuer : ajouter, modifier une colonne.

Ajouter une colonne syntaxe :

L'ajout d'une colonne dans une table est généralement très simple et peut se faire grâce à l'instruction SQL telle que celle-ci :

```
ALTER TABLE nom_table
ADD nom_colonne type_données
```

Exemple : Pour ajouter une colonne qui correspond à une rue à une table utilisateur, il est possible d'utiliser la requête suivante :

```
ALTER TABLE utilisateur
ADD adresse_rue VARCHAR (255)
```

3. LA COMMANDE SELECT

L'utilisation la plus la plus courante du SQL consiste à lire les données provenant de la base de données. Cela s'effectue grâce à la commande SELECT qui retourne des enregistrements dans un tableau de résultats. Cette colonne peut sélectionner une ou plusieurs colonnes d'une table.

Exemple :

```
SELECT nom_du_champ
FROM nom_de_la_table ;
```

Cette requête va sélectionner (SELECT) le champ « nom_du_champ » provenant FROM de la table appélée « nom_de_la_table ».

Imaginons que nous avons une table appelée « clients » et contenant les informations sur les clients de l'entreprise.

Table « **clients** »

IDENTIFIANT	PRENOM	NOM	COMMUNE
1	Valerie	Nawen	Lemba
2	Vanessa	messa	Mont-Ngafula
3	Stéphanie	zonyo	Lemba
4	Odette	nyalota	Limete

Si l'on veut avoir la liste des communes de nos clients, nous allons faire la requête suivante :

SELECT commune
FROM clients ;

Résultat

COMMUNE
Lemba
Mont-Ngafula
Lemba
Limete

Avec la même table client, il est possible de lire plusieurs colonnes à la fois. Il suffit tout simplement de séparer les noms de champs souhaités par une virgule. Pour obtenir les prénoms et les noms des clients, il faut alors faire la requête :

```
SELECT prenom, nom
FROM clients;
```

A VOUS DE DEVINER LES RESULTATS

Il est possible de retourner toutes les colonnes d'une table sans avoir besoin de connaitre les noms de ces colonnes. Au lieu de lister toutes les colonnes, il faut simplement utiliser le caractère « * » (étoile). C'est un joker qui permet de sélectionner toutes les colonnes. Il s'utilise de la manière suivante :

```
SELECT *FROM clients ;
```

Cette requête va retourner exactement les mêmes colonnes se trouvant dans la base de données. Dans notre cas, le résultat sera donc :

IDENTIFIANT	PRENOM	NOM	COMMUNE
1	Valerie	Nawen	Lemba
2	vanessa	Messa	Mont-Ngafula
3	stéphanie	Zonyo	Lemba
3	Odette	Nyalota	Limete

Avec la requête SELECT, on peut :

- Joindre une autre table aux résultats ;
- Filtrer pour ne sélectionner que certains résultats ;
- Classer les résultats ;
- Grouper les résultats pour faire uniquement des statistiques (note moyenne, prix le plus élevé…)

Une requête SELECT peut devenir plus assez longue. Juste à titre informatif, voici une requête SELECT qui possède presque toutes les commandes possibles :

```
SELECT *
FROM table
WHERE condition
GROUP BY expression
HAVING condition
{UNION I INTERSECT I EXCEPT}
ORDER BY expression
LIMIT count
OFFSET start
```

4. COMMANDE DISTINCT

L'utilisation de la commande SELECT en SQL permet de lire toutes les données d'une ou plusieurs colonnes. Cette commande peut potentiellement afficher les doublons, alors pour éviter la redondance dans le résultat, il est conseillé d'ajouter simplement DISTINCT après le mot SELECT.

```
SELECT DISTINCT ma_colonne
```

```
FROM nom_de_la_table ;
```

Cette requête sélectionne le champ « **ma_colonne** » de la table « **nom_de_la_table** » en évitant de retourner des doublons.

Pour notre cas de la table clients :

En utilisant seulement SELECT tous les noms sont retournés, or la table contient plusieurs fois le même prénom. Pour sélectionner uniquement les prénoms uniques il faut utiliser la requête suivante :

```
SELECT DISTINCT prenom
FROM clients;
```

5. **ALIAS AS**

Dans le langage SQL il est possible d'utiliser des ALIAS pour renommer temporairement une colonne ou une table dans une requête. Cette astuce est particulièrement utile pour faciliter la lecture des requêtes. Il permet de renommer le nom d'une colonne dans les résultats de la requête SQL. C'est pratique pour avoir un nom facilement identifiable dans une application qui doit ensuite exploiter les résultats d'une recherche.

Par exemple, pour renommer une colonne de la **colonne1** à **C1** la requête est la suivante :

```
SELECT colonne1 AS C1, Colonne2
FROM clients ;
```

Cette syntaxe peut également s'afficher de la façon suivante :
SELECT colonne1 C1, colonne2
FROM clients;

LES REQUETES SIMPLES

6. LA COMMANDE WHERE

La commande WHERE dans une requête SQL permet d'extraire les lignes d'une base de données qui respectent une condition donnée. Cette requête permet d'obtenir uniquement les informations désirées.

Syntaxe

La commande **WHERE** s'utilise en complément à une requête utilisant **SELECT**. La façon la plus simple de l'utiliser est la suivante :

```
SELECT nom_colonnes
FROM nom_table
WHERE condition ;
```

Exemple : imaginons une base de données "client" qui contient le nom des clients, le nombre de commandes qu'ils ont effectuées et leurs villes.

id	nom	Nbr_commande	ville
1	Valérie	3	Lubumbashi
2	Vanessa	0	Kinsahsa
3	Stéphanie	1	Kolwezi

Pour obtenir seulement la liste des clients qui habitent à Kolwezi, il faut taper la requête suivante :

```
SELECT *FROM client
WHERE ville ='Kinshasa';
```

7. LES OPERATEURS AND et OR

Une requête SQL peut être restreinte à l'aide de la condition WHERE. Les opérateurs logiques AND et OR peuvent être utilisés au sein de la commande WHERE pour combiner des conditions. Les opérateurs sont ajoutés dans la condition WHERE. Ils peuvent être combinés à l'infini pour filtrer les données comme souhaités.
L'opérateur AND permet de s'assurer que la condition1 et la condition2 sont vraies.

```
SELECT nom_colonnes
FROM nom_table
WHERE condition1 AND condition2;
```

L'opérateur OR vérifie quant à lui que la condition1 ou la conditions2 est vraie :

```
SELECT nom_colonnes
FROM nom_table
WHERE condition1 OR condition2;
```

Ces opérateurs peuvent être combinés à l'infini. L'exemple ci-dessous filtre les résultats de la table « nom_table » si condition1 et condition2 ou condition3 sont vraies :

```
SELECT nom_colonnes FROM no_table
WHERE condition1 AND (condition2 OR condition3);
```

Pour illustrer les prochaines commandes, nous allons considérer la table "produit" suivante :

Id	Nom	Catégorie	Stock	Prix
1	ordinateur	Informatique	6	950
2	clavier	Informatique	23	35
3	souris	Informatique	16	20
4	crayon	fourniture	147	2

L'opérateur AND permet de joindre plusieurs conditions dans une requête. En gardant la même table que précédemment, pour filtrer uniquement les produits informatiques qui sont presque en rupture de stock (moins de 20 produits disponibles) il faut exécuter la requête suivante :

```
SELECT *FROM produit
WHERE catégorie='informatique' AND stock < 20;
```

Pour filtrer les données afin d'avoir uniquement les informations sur les produits « ordinateur » ou clavier, il faut effectuer la recherche suivante :

```
SELECT * FROM produit
WHERE nom='ordinteur' OR nom='clavier';
```

8. LA COMMANDE IN

L'opérateur logique IN dans SQL s'utilise avec la commande WHERE pour vérifier si une colonne est égale à une des valeurs comprises dans set des 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 'valeur1' ou 'valeur2' ou 'valeur3'. Il est possible d'utiliser la syntaxe suivante :

```
SELECT nom_colonne
FROM nom_table
WHERE nom_colonne IN ('valeur1', 'valeur2', 'valeur3'…) ;
```

A savoir, entre les parenthèses il n'y a pas de limites du nombre d'arguments. Il est possible d'ajouter encore d'autres valeurs.

La syntaxe utilisée avec l'opérateur IN est plus simple que d'utiliser une succession d'opérateur OR. Pour le montrer concrètement avec un exemple, voici deux requêtes qui retourneront les mêmes résultats. Dans l'une, nous utiliserons l'opérateur IN et dans l'autre nous utiliserons l'opérateur OR.

Requête avec l'opérateur OR

```
SELECT *FROM utilisateur
WHERE prenom='Vanessa' OR prenom=' Valerie' OR
prenom='Stéphanie';
```

Requete avec l'opérateur IN

```
SELECT * FROM utilisateur
WHERE prenom IN ('Vanessa',Valerie', 'Stéphanie')
```

9. LA COMMANDE 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 chaine de caractères, des nombres ou des dates. L'exemple le plus illustratif consiste à récupérer seulement uniquement les enregistrements entre deux dates prédéfinies.

Syntaxe :

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

```
SELECT *
FROM table
WHERE nom_colonne BETWEEN 'valeur1' AND 'valeur2';
```

10. LA COMMANDE LIKE

L'opérateur LIKE est utilisé dans la clause WHERE des requêtes SQL et permet d'effectuer une recherche sur un modèle particulier. Il est par exemple facile de rechercher les enregistrements dont la valeur d'une colonne commence par telle ou telle lettre. Les modèles de recherche sont multiples. La syntaxe à utiliser pour faire usage de l'opérateur LIKE est la suivante :

```
SELECT * FROM table colonne LIKE modèle.
```

Explications de la syntaxe

- **LIKE '%a% :** le caractère modulo 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 « a »
- **LIKE 'a%' :** ce modèle permet de rechercher les lignes de « colonne » qui commence par « a ».
- **LIKE '%a%'** : ce modèle est utilisé pour rechercher tous les enregistrements qui utilisent le caractère « a ».
- **LIKE 'pa%on' :** ce modèle permet de rechercher tous les modèles qui commencent par « pa » et se terminent par « on ».

Illustration

Si l'on souhaite obtenir seulement les clients des villes donnent les noms commencent par N, la requête sera la suivante :

```
SELECT*
FROM clients
WHERE villle LIKE 'N%';
```

11. LA COMMANDE IS NULL / IS NOT NULL

Dans le langage SQL, l'opérateur IS permet de filtrer les résultats qui contiennent la valeur NULL. Cet opérateur est indispensable car la valeur NULL est une valeur inconnue et ne peut par conséquent pas être filtrée par les opérateurs de comparaison. Pour filtrer les résultats où les champs d'une colonne sont à NULL, il convient d'utiliser la syntaxe suivante :

```
SELECT *
FROM 'table'
WHERE nom_colonne IS NULL;
```

A l'inverse pour filtrer les résultats et obtenir uniquement les enregistrements qui ne sont pas NULL, il convient d'utiliser la syntaxe suivante :

```
SELECT *
FROM 'table'
WHERE nom_tableIS NOT NULL;
```

12. LA COMMANDE GROUP BY

La commande GROUP BY est utilisé dans SQL pour grouper plusieurs résultats et utiliser une fonction des totaux sur un groupe des résultats. Sur une table qui contient toutes les ventes d'un magasin, il est par exemple possible de lister, de regrouper les ventes par clients identiques et obtenir le cout total des achats pour chaque client.

De façon générale, la commande GROUP BY s'utilise de la façon suivante :

```
SELECT colonne1, Fonction(colonne2)
FROM table
GROUPE BY colonne1
```

Id	client	tarif	date
1	Valérie	200	2018-02-10
2	stéphanie	100	2018-03-10
3	vanessa	300	2013-06-08

4	simeon	25	2012-06-17

```
SELECT client, SUM(tarif)
FROM achat
GROUP BY client ;
```

La fonction **SUM ()** permet d'additionner la valeur de chaque tarif pour un même client. La commande GROUP BY se fait très souvent accompagnée des fonctions d'agrégations statistiques, les principales sont les suivantes :

- AVG () : pour calculer la moyenne d'un set de valeur. Permet de connaitre le prix du panier moyen de chaque client.
- COUNT () : pour compter le nombre des lignes concernées. Permet de compter le nombre d'achat effectuer par chaque client.
- MAX () : pour récupérer la plus haute valeur. Pratique pour savoir l'achat le plus chèr.
- MIN () : Pour récupérer la plus petite valeur. Utile par ex pour connaitre la date d'achat du premier client.
- SUM () : pour calculer la somme de plusieurs lignes. Permet par exemple le total de tous les achats d'un client. Ces petites fonctions se révèlent rapidement importantes pour travailler sur les données.

13. LA COMMANDE HAVING

La condition HAVING en SQL est presque similaire à WHERE à la seule différence que HAVING permet de filtrer en utilisant des fonctions telles que SUM (), COUNT (), MIN () ou MAX ().
On utilise la commande HAVING de la façon suivante :

```
SELECT colonne1, SUM(colonne2)
FROM nom_table
GROUP BY colonne1
HAVING function(colonne2) opérateur valeur
```

14. LA COMMANDE ORDER BY

La commande ORDER BY permet de trier les lignes dans un résultat d'une requête SQL. Il est possible de trier les données sur une ou plusieurs colonnes, par ordre ascendant ou descendant.

Syntaxe

Une requête où l'on souhaite filtrer l'ordre des résultats utilise la commande ORDER BY de la sorte :

```
SELECT colonne1, colonne2
FROM table
ORDER BY colonne1;
```

Par défaut, les résultats sont classés par ordre ascendant, toutefois il est possible d'inverser l'ordre en utilisant le suffixe DESC après le nom de la colonne. Par ailleurs, il est possible de trier sur plusieurs colonnes en les séparant par une virgule. Une requête plus élaborée ressemblerait à ceci :

```
SELECT colonne1, colonne2, colonne3
FROM table
ORDER BY colonne1 DESC, colonne2 ASC;
```

NB : Ce n'est pas une obligation d'utiliser le suffixe ASC sachant que les résultats s'affichent par défaut en ordre ascendant.

Alors, s'il faut par exemple récupérer la liste des utilisateurs par ordre alphabétique du nom de famille, il est possible d'utiliser la requête suivante :

```
SELECT *
FROM utilisateur
ORDER BY nom
```

15. LA COMMANDE UNION

La commande UNION de SQL permet de mettre bout-à-bout les résultats des plusieurs requêtes utilisant elles-mêmes la commande SELECT. C'est donc une commande qui permet de concaténer les résultats de deux requêtes ou plus.

La syntaxe pour unir les résultats de 2 tableaux sans afficher les doublons est la suivante :

```
SELECT * FROM table1
UNION
SELECT * FROM table2
```

16. LA COMMANDE INTERSECT

La commande SQL INSERSECT permet d'obtenir l'intersection des résultats de 2 requêtes. Cela peut s'avérer utile lorsqu'il faut trouver s'il y a des données similaires sur 2 tables distinctes. Pour l'utiliser convenablement, il faut que les deux tables (requêtes) utilisent le même nombre de colonnes, avec le même type et dans le même ordre.
La syntaxe à adopter pour utiliser cette commande est la suivante :

```
SELECT *FROM table1
INTERSECT
SELECT* FROM table2
```

17. LA COMMANDE INSERT INTO

L'insertion des données dans une table s'effectue à l'aide de la commande INSERT INTO. Cette commande permet au choix d'inclure une seule ligne à la base existante ou plusieurs lignes d'un coup.

Insertion d'une ligne à la fois

Pour insérer les données dans une base, il y a deux syntaxes qui s'offrent à nous :

- Insérer une ligne en indiquant les informations pour chaque colonne existante (en respectant l'ordre).
- Insérer une ligne en spécifiant les colonnes que vous souhaitez compléter. Il est possible d'insérer une ligne et en renseigner seulement une partie des colonnes.

La syntaxe pour remplir une ligne avec cette méthode est la suivante :

```
INSERT INTO table
VALUES ('valeur1', 'valeur2', 'valeur'…)
```

Cette syntaxe possède les avantages et les inconvénients suivants :

- Obligation de remplir toutes les données, tout en respectant l'ordre des colonnes
- Il n'y a pas le nom des colonnes, donc les fautes de frappe sont très limitées.
- Par ailleurs, les colonnes peuvent être limitées sans avoir à changer la requête.
- L'ordre des colonnes doit être identique si non, certaines valeurs prennent le risque d'être complétées dans la mauvaise colonne. Cette énième solution est très similaire, excepté qu'il faut indiquer le nom des colonnes avant « VALUES ». La syntaxe est la suivante :

```
INSERT INTO table (nom_colonne1, nom_colonne2, …)
VALUES ('valeur1', 'valeur2'…) ;
```

A noter que lorsque le champ à remplir est du type VARCHAR ou TEXT, la valeur doit être obligatoirement mise entre guillemets. En revanche, lorsque la colonne est un numérique tel que INT ou BIGINT, il n'y a pas nécessité d'utiliser les guillemets, il suffit juste d'identifier le nombre.

5. LA COMMANDE UPDATE

La commande UPDATE permet d'effectuer les modifications sur les lignes existantes. Très souvent cette commande est utilisée avec WHERE pour spécifier sur quelles lignes doivent porter la ou les modifications.

Syntaxe :
La syntaxe basique d'une requête utilisant UPDATE est la suivante :

```
UPDATE table
SET nom_colonne1='nouvelle_valeur'
WHERE condition
```

Cette commande permet d'attribuer une nouvelle valeur à la colonne nom_colonne_1 pour les lignes qui respectent les conditions stipulées par WHERE. Il est aussi possible la même valeur à la colonne nom_colonne_1 pour toutes les lignes d'une table si la condition WHERE n'était pas spécifiée.

```
UPDATE etudiants
SET nom='BOKUNGU', ville='Kinshasa', profession='étudiant'
WHERE id=2 ;
```

6. LA COMMANDE DELETE

La commande DELETE en SQL permet de supprimer les lignes dans une table. En utilisant cette commande associée à WHERE, il est possible de sélectionner les lignes concernées qui seront supprimées.

Attention, avant d'essayer de supprimer les lignes, il est recommandé d'effectuer une sauvegarde de la base de données, ou tout du moins de la table concernée par la suppression. Ainsi, s'il y a eu une mauvaise manipulation, il est toujours possible de restaurer les données.

Syntaxe :

La syntaxe pour supprimer les lignes est la suivante :

```
DELETE FROM table
WHERE condition
```

Attention, s'il n'y a pas de condition WHERE, toute les lignes seront alors supprimées et la table sera vide.

```
DELETE FROM utilisateurs
WHERE date_inscription <  '2012-04-10' ;
```

TABLE DES MATIERES

REFERENCES

I. NOTES DE COURS

- Mabela Rostin, Cours de base de données G3 AIA, UNIKIN, 2019.
- Franck Leroy, notes de cours de base de données, Ecole du commerce, Rênes, ed. laFac, 2014 ;
- Kutangila David, Notes de cours de méthodes d'analyse informatique G2 AIA, UNIKIN, 2018.

II. WEBOGRAPHIE

- **www.udemy.com/base_de_donnees/sgbd/relationnel, consulté le 12 Décembre2 2025.**
- **www.site_du_zero.com, consulté le 12/ 01 / 2023**