# BASE DE DONNÉES

# INTRODUCTION

# BASE DE DONNÉES

## Introduction

Une base de donnée est un ensemble d'informations qui est organisé de manière à être facilement accessible et exploitable par les systèmes de traitement.

Elle est utilisée comme méthode de stockage, de gestion, de récupération et de mise à jour de l'information.

Il s'agit d'un ensemble de données structurées et mémorisées sur un support de stockage persistant.

L'objectif d'une base de donnée est d'enregistrer des faits, des opérations et des informations au sein d'un organisme, il s'agit d'un élément essentiel de tout système informatique qui permettra d'opérer des schémas de traitement non linéaires.

# Introduction

Une base de donnée a en partie pour objectif de fournir une structure de rangement de la donnée pour rendre exploitable et compatible le traitement qui va être opéré à partir de celle-ci.

Exemple : Fichier CSV d'utilisateurs

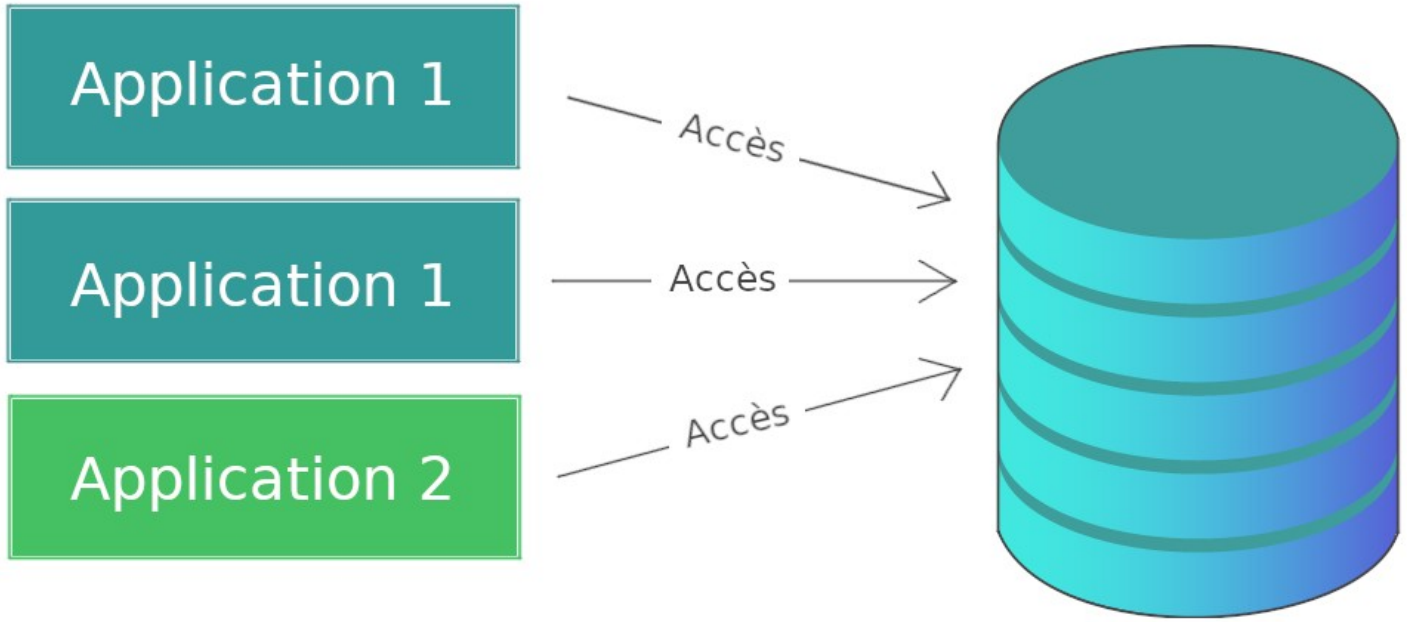
```
"Username" ; "Password" ; "MailAddress" ; "FirstName" ; "LastName"
"ttoto" ; "012345" ; "toto@toto.toto" ; "toto" ; "toto"
"jdoe" ; "password" ; "john.doe@toto.local" ; "John" ; "DOE"
"admin" ; "admin" ; "admin@toto.local" ; "admin" ; ""
"alexandre" ; "L0urDM0tDeP@553" ; "arobine@toto.local" ; "Alexandre " ; "ROBINE"
"jdive" ; "Oce#AERergh!123" ; "jdive@toto.local" ; "Jean" ; "DIVE"
```

Il existe cependant plusieurs problématiques à un traitement sur la base d'un fichier tel que celui-ci.

## Introduction

La première de ces problématiques est l'accès concurrent à la donnée.

En effet, un fonctionnement sur la base d'un fichier ne permet pas de gérer les besoins des différentes applications en simultané (c'est également le cas pour une même application opérant plusieurs accès à la même donnée en parallèle).



## Introduction

La mise en place d'un système de gestion de base de données (SGBD) permet de gérer les accès concurrent mais également la séparation des données et des applications.

En travaillant avec un espace de stockage en fichiers, les données appartiennent à un programme. La base de donnée permet de les en sortir et ainsi travailler avec des données autonomes hors des applications.

Le travail en fichier entraîne généralement une redondance pour des raisons opérationnelles entraînant une grande difficulté de mise à jour. La base de donnée permet donc de travailler sur ces problématiques de partage de données.

Il s'agira également d'un point essentiel lorsqu'il sera question de sauvegarder ces données.

# Introduction

Notre SGBD doit permettre de :

- Décrire les données : Cette description s'opère indépendamment des applications au travers d'un « Data Definition Language (DDL) »

- Manipuler les données : Il doit être possible d'interroger et mettre à jour les données sans préciser d'algorithme d'accès au travers d'un langage de requêtes déclaratif. On parle de DML ou « Data Manipulation Language »

- Contrôler les données : Il est nécessaire de préserver la confidentialité de la donnéeau travers d'un système de contrôle d'accès et d'autorisations mais aussi l'intégrité de celles-ci. On parle de DCL « Data Control Language »

## Introduction

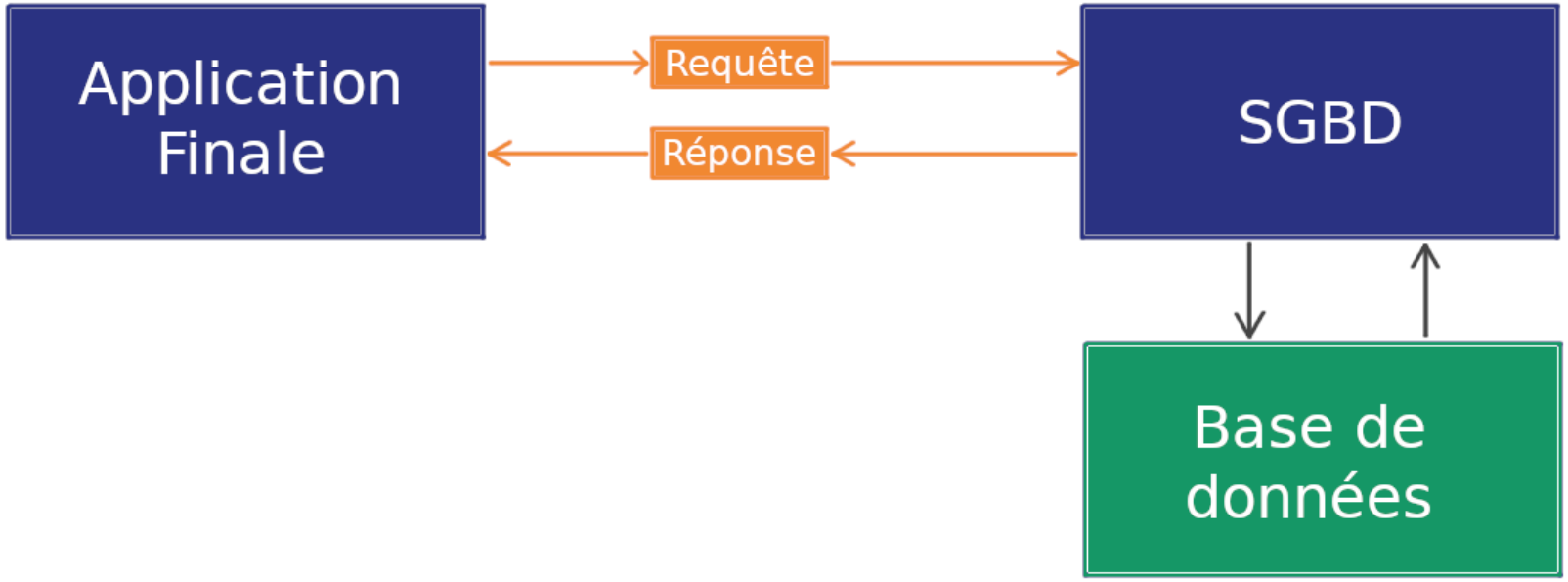
En dernier lieu, notre SGBD permet de garantir une indépendance à la fois logique et physique.

Indépendance logique : Permettre aux applications d'avoir une vue différente de la même donnée. Permettre une normalisation des schémas logiques des bases de données sans pour autant causer un impact de production au niveau des schémas applicatifs les exploitant.

Indépendance physique : Permette la modification de la structure de stockage (disque, méthode d'accès, mode de placement, encodage, …) sans causer un impact de production au niveau des schémas applicatifs les exploitant.

## Introduction

L'idée derrière notre SGBD est de fournir une couche d'abstraction afin que notre application exploitant la donnée ne travaille pas directement sur la donnée mais bien au travers d'un système de requêtes/réponses vers un système manipulant les données en fonction des besoins applicatifs.

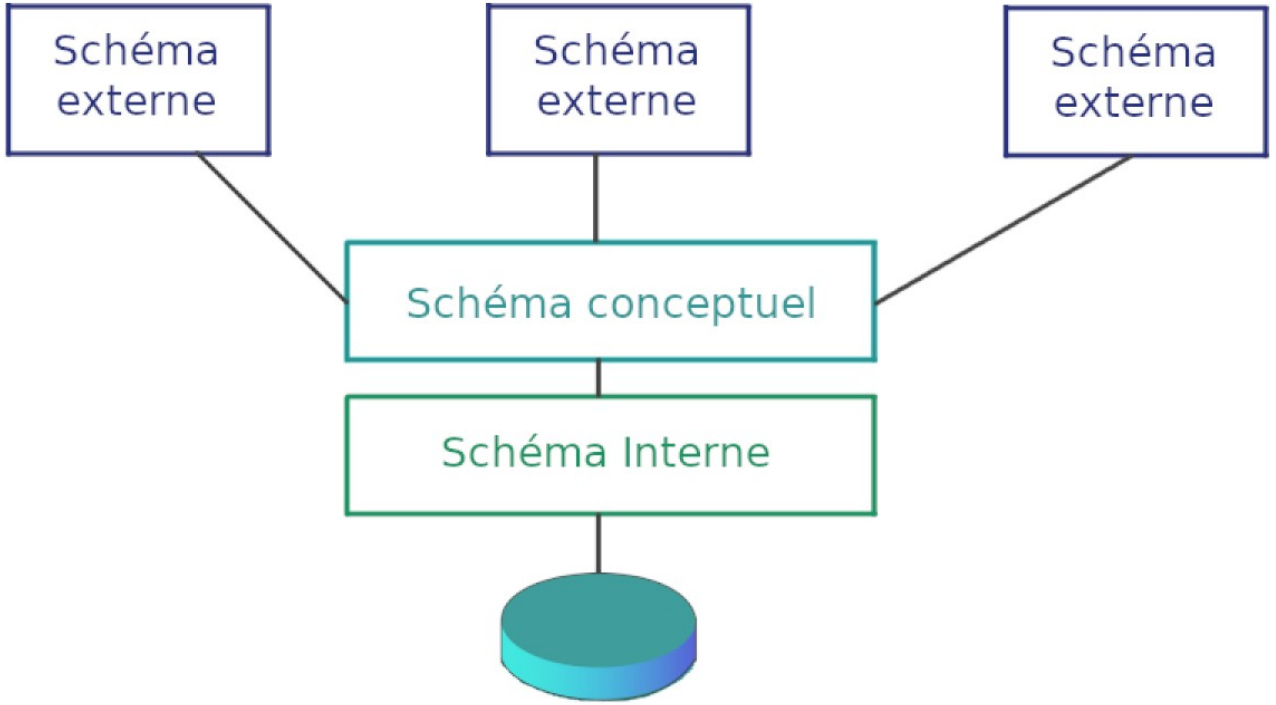


# ARCHITECTURE ET MODÉLISATION

## Architecture et modélisation

L'architecture ANSI/SPARC est un modèle proposé en 1975. Il s'agit de l'architecture fondamentale sur laquelle repose nos SGBD modernes. Cette architecture est divisée en 3 niveaux :

- Schéma interne (SI)
- Schéma conceptuel (SC)
- Schéma externe (SE)



## Architecture et modélisation

Schéma externe :

Le concept de « vue » permet d'obtenir l'indépendance logique recherchée dans la conception de notre SGBD.

La modification du schéma logique n'entraîne pas la modification des applications (uniquement une modification de la vue).

Chaque vue correspond à la perception d'une partie des données mais aussi des données synthétisées à partir des données en bases (statistiques, ...)

# Architecture et modélisation

## Schéma conceptuel:

Ce niveau contient la description des données et des contraintes d'intégrité. La base est définie et manipulée à ce niveau sans avoir à se soucier de l'implémentation basse de la donnée.

## Schéma Interne :

Le schéma interne correspond aux structures de stockage et aux moyens d'accéder à la donnée (index, ...)

## Architecture et modélisation

Dans le modèle d'implémentation de nos SGBD, on distingue différents types de base de données :

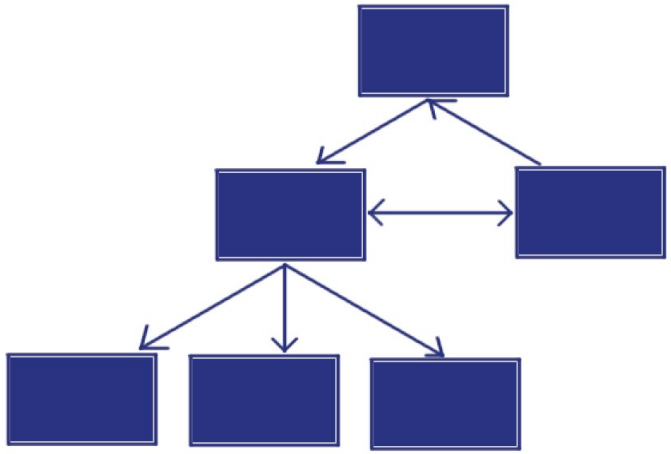
- Base de données Réseau
- Base de données hiérarchique
- Base de données Objet
- Base de données relationnelles
- Base de données NoSQL

# Architecture et modélisation

## Base de données réseau

Les bases de données réseaux sont un type de modèle dans lequel plusieurs enregistrements ou fichiers membres peuvent être liés à plusieurs fichiers propriétaires et vice-versa. Le modèle peut être vu comme un arbre où chaque information sur les membres est la branche liée au propriétaire.

Essentiellement, les éléments sont sous la forme de réseau ou un seul élément peut pointer vers plusieurs éléments de données et peut lui même être pointé par plusieurs éléments.

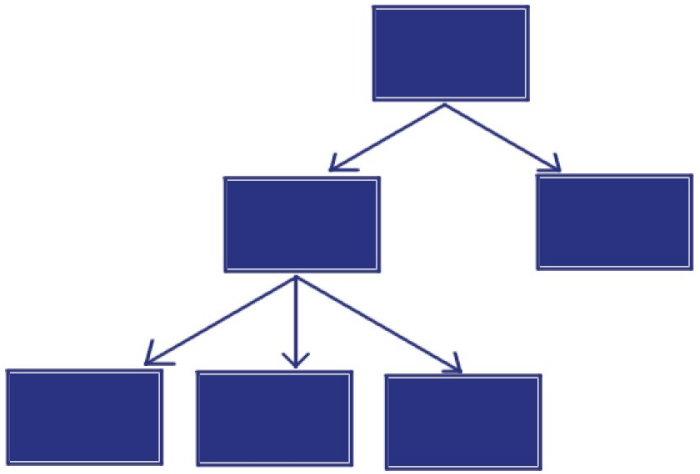


## Architecture et modélisation

### Base de données hiérarchique

Les bases de données hiérarchiques sont un type de base de données organisées sous la forme de plusieurs arbres avec des branches. Avec cette arborescence, chaque élément dépend d'un seul élément parent. Plusieurs éléments peuvent dépendre du même élément parent.

Il s'agit d'une version similaire (plus spécifique) d'une base de donnée réseau.



## Architecture et modélisation

### Base de données objet

Dans une base de données de type objet, les éléments sont stockés sous la forme d'objets (de classes) qui vont elles-mêmes contenir des données membres.

Les champs au sein de la base sont des instances de ces classes. Ce mode d'implémentation permet de d'enregistrer et de rechercher rapidement et simplement des ensembles de données complexes.

Chaque objet possède un identifiant unique automatiquement attribué. L'objectif est d'offrir une très forte compatibilité avec les langages de développement orientés objet (même si ce type de base est très peu répandu et assez peu performante sur de trop grands ensembles).

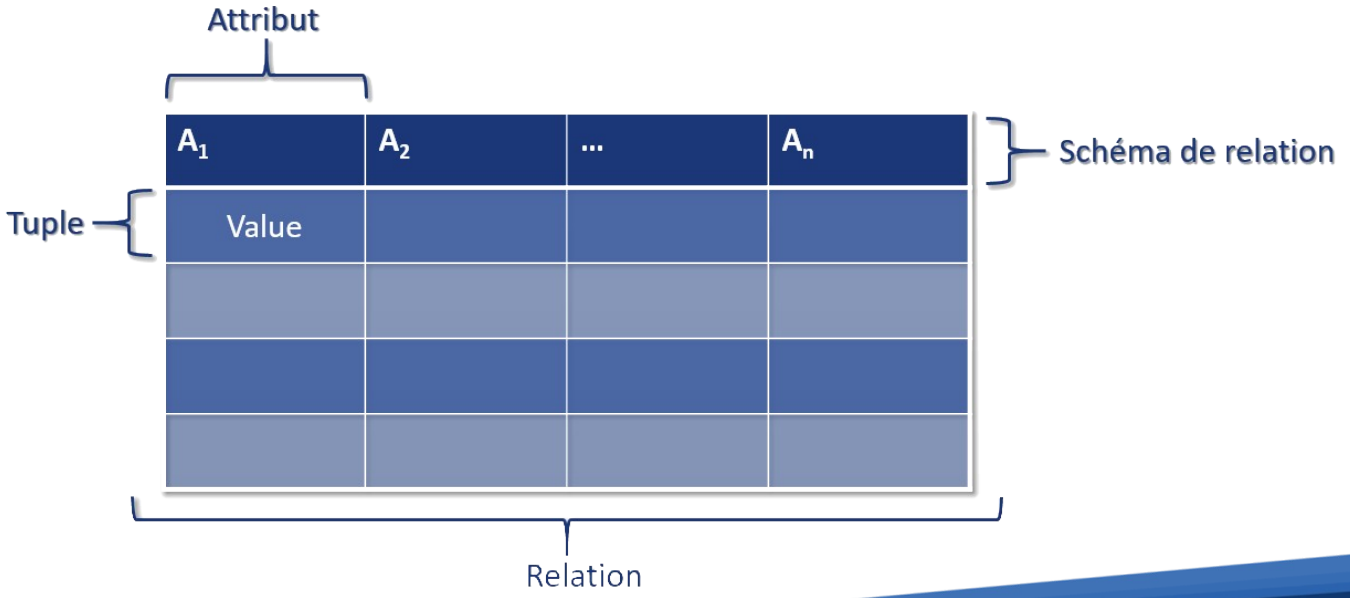
# Architecture et modélisation

## Base de données relationnelles

La base de donnée relationnelle repose sur un schéma de « relations », une relation représente un ensemble d'éléments possédant les mêmes propriétés.

Chaque relation est constituée d'une série d'enregistrements de données (appelés tuple) dont les valeur sont affectés à certains attributs.

Il s'agit du modèle d'organisation le plus fréquemment rencontré.



# Architecture et modélisation

## Base de données NoSQL

Une base de données NoSQL (Not Only SQL) est une base de donnée « non relationnelle » . Il est possible d'y stocker des données sous une forme non structurée ne possédant pas de schéma fixe.

Les liens entre les données ne sont ainsi plus nécessaire ce qui facilite un rangement à très grande échelle. Ce type de base de données a été introduit pour faciliter la gestion des données volumineuses ainsi que les données qui nécessitent une grande flexibilité (parsing dynamique d'une donnée selon un schéma changeant).

Il existe plusieurs type de base de données NoSQL (paire clé/valeur, document, colonnes, graph).

# STRUCTURED QUERY LANGUAGE (SQL)

## Structured Query Language (SQL)

SQL (Structured Query Language) est un langage standardisé dans les années 70 pour gérer des données au sein des bases de données relationnelles.

Au sein des systèmes de gestion de base de données relationnelles (SGBDR), il s'agit du langage quasi universellement utilisé.

Le langage SQL possède la même limite que le SGBDR, il a une capacité très limité à traiter des données complexes (comme des données non structurées).

SQL est composé d'un langage de contrôle de données (LCD), d'un langage de définition des données (LDD), et d'un langage de manipulation de données (LMD) et pour finir d'un langage de contrôle des transactions (LCT). Ces langages sont des sous-ensembles constituant le langage SQL.

## Structured Query Language (SQL)

Pour rappel :

Le LCD (Langage de contrôle de données) gère les utilisateurs et les privilèges d'une base de données

Le LDD (Langage de définition de données) gère les créations et les suppressions des objets d'une base de données

Le LMD (Langage de manipulation de données) gère la manipulation des données stockées dans les tables d'une base de données (manipuler une ligne, …)

Le LCT (Langage de contrôle des transactions) permet la validation et l'annulation des modifications des éléments d'une base de données

## Structured Query Language (SQL)

Chaque sous langage structurant le langage SQL est lié à un jeu d'instructions permettant de créer une transaction.

Une transaction est un ensemble d'ordres SQL.

Le but d'une transaction est de permettre la mise à jour d'une base de données en garantissant la cohérence des états successifs de celle-ci.

| LCT<br><br>Set transaction, Commit, Rollback | | |
|---|---|---|
| LDD<br><br>Create<br>Alter<br>Drop | LMD<br><br>Insert<br>Update<br>Delete<br>Select | LCD<br><br>Connect<br>Grant<br>Revoke |

## Structured Query Language (SQL)

Chaque transaction SQL garantie les propriétés ACID. Les propriétés ACID (Atomicité, Cohérence, Isolation, Durabilité) sont un ensemble de propriétés qui garantissent qu'une transaction informatique quelconque est exécutée de façon fiable :

- Atomicité : Les mises à jour doivent être atomiques (Elles doivent être totalement réalisées ou pas du tout).

- Cohérence : Les modifications apportées à la base doivent être valides (Elles doivent respecter le schéma d'implémentation de la base de données)

- Isolation : Les transactions simultanées ne doivent jamais interférer entre elles

- Durabilité : Toutes les transactions sont lancées de manière définitive

# Structured Query Language (SQL)

## Syntaxe d'une transaction :

*SET autocommit = 0 ;*
*BEGIN TRANSACTION ;*

*Ordre 1 ;*
*.....*
*Ordre n ;*
*Commit ;*

Chaque instruction doit se terminer par un ; bornant celle-ci.

## Transaction Control Language (TCL)

Commit : Permet la validation des modifications d'une donnée

Rollback : Annule la transaction en cours

Les données sont restituées dans l'état avant le début de la transaction.

Toutes les ressources utilisées par la transaction sont également libérées.

# Structured Query Language (SQL)

Au sein d'un serveur de base de données SQL, il est possible de rencontrer plusieurs base de données hébergées par un même système.

Il est donc possible de lister ces bases :

*SHOW DATABASES ;*

Et de sélectionner celle qui nous intéresse :

*USE specific_db ;*

Ces instructions, comme les autres doivent être bornées par un ;

# Structured Query Language (SQL)

## Data Definition Language (DDL)

### Création d'une base de données :

*CREATE DATABASE [IF NOT EXISTS] specific_db ;*

### Suppression d'une base de données :

*DROP DATABASE [IF EXISTS] specific_db ;*

La suppression d'une base de données implique la suppression de l'ensemble des tables, des vues et des données.

# Structured Query Language (SQL)

## Création d'une table

```
CREATE TABLE nom_de_la_table
(
   colonne1 type_donnees [contrainte],
   colonne2 type_donnees [contrainte],
   colonne3 type_donnees [contrainte],
   colonne4 type_donnees [contrainte]
)
```

Pour définir une colonne à la création d'une table, il faut spécifier :

- Un nom d'attribut
- Un type
- Les contraintes applicables

Les contraintes sont les règles applicables aux colonnes de donnée d'une table pour gérer les données pouvant êtres insérées dans cette table.

Dans le but de garantir l'exactitude et la fiabilité des données en base.

## Structured Query Language (SQL)

Une contrainte peut s'appliquer de deux manières différentes :

- Contrainte de colonne : qui ne s'applique que sur la colonne
- Contrainte de table : qui s'applique sur l'ensemble de la table

Il existe un certain nombre de types de contraintes :

- NOT NULL : Interdit l'insertion d'une valeur NULL pour l'attribut
- UNIQUE : Deux tuples (rows) ne peuvent pas recevoir des valeurs identiques sauf NULL
- PRIMARY KEY : Désigne la clé primaire de la table (attribut unique)
- FOREIGN KEY : Contrainte d'intégrité utilisée pour regrouper deux tables
- DEFAULT value : Permet de spécifier la valeur par défaut d'un attribut
- .....

# Structured Query Language (SQL)

## Exemple : Création d'une table avec contraintes :

```
CREATE TABLE toto
(
   toto_id INT PRIMARY KEY,
   prenom VARCHAR (50) NOT NULL,
   nom_famille VARCHAR (50) NOT NULL,
   age DEC CHECK(age > 0)
) ;
```

# Structured Query Language (SQL)

Le schéma de nos données peut évoluer au cours de la production au niveau de notre base de données, pour ce faire, il est possible d'altérer les tables présentes :

## Ajout d'une colonne

*ALTER TABLE table1 ADD nom_colonne type_donnee contrainte*

## Suppression d'une colonne

*ALTER TABLE table1 DROP nom_colonne*

## Modification d'une colonne (changement de type d'une colonne, des contraintes, …)

*ALTER TABLE table1 MODIFY nom_colonne type_donnee*

L'objectif est d'avoir un schéma évolutif et hautement compatible avec les besoins réels.

## Structured Query Language (SQL)

Dans le langage SQL, il existe un champ permettant d'identifier de manière unique chaque enregistrement dans une table de base de donnée, il s'agit de la primary key (la clé primaire).

Chaque enregistrement de celle clé primaire doit être unique et ne doit pas contenir de valeur NULL. Cette clé sert d'index, chacune des tables ne peut contenir qu'une seule clé primaire.

L'usage le plus fréquent consiste à créer une colonne numérique qui s'incrémente automatiquement à chaque enregistrement (grâte à AUTO_INCREMENT)

| créer | supprimer |
| --- | --- |
| ALTER TABLE table1 ADD PRIMARY KEY (id) ; | ALTER TABLE table1 DROP PRIMARY KEY ; |

## Structured Query Language (SQL)

Il existe également des clés étrangères (foreign key) qui permet de gérer les relations entre deux tables. De la même manière, cette clé doit être unique et non nulle.

Celle-ci sert de contrainte qui assure un référentiel d'intégrité au sein d'une base de données SQL.

L'idée est d'utiliser une colonne (ou un ensemble de colonnes) pour établir un lien entre les données de deux tables.

Pour créer une contrainte de clé étrangère, la table parent doit posséder une clé primaire.

| créer | supprimer |
|---|---|
| ALTER TABLE table1 ADD CONSTRAINT ForeignK FOREIGN KEY (table2) REFERENCES table2(id) ; | ALTER TABLE table1 DROP FOREIGN KEY ForeignK ; |

## Structured Query Language (SQL)

Enfin, il est possible d'associer des index au sein de notre système SQL. Un index fonctionne sur le même principe que l'index d'un livre. Cet index va permettre au système de notre base de données de rechercher les données sur la base de celui-ci.

Même si ces index occupent de l'espace supplémentaire au sein de notre base, la recherche dans la base sera optimisée. Cependant, l'insertion de données sera plus long car les index sont mis à jour à chaque fois qu'une donnée est insérée en base.

Sur une base possédant des milliers voire des millions d'enregistrement, l'utilisation d'index permet de gagner un temps considérable pour la lecture des données

| créer | supprimer |
| --- | --- |
| CREATE INDEX index1 ON table1 (colonne1, colonne1, …) ; | ALTER TABLE table1 DROP INDEX index1 ; |

## Structured Query Language (SQL)

Pour en finir avec le cycle de vie de notre table et des données qu'elle héberge, il est possible de la supprimer facilement.

*DROP TABLE [IF EXISTS] table_name [, TABLE_NAME] …*

Au milieu de ce cycle de vie, il est nécessaire d'insérer des tuples dans nos tables. Il est possible d'en insérer plusieurs à la fois.

*INSERT INTO table1(col1, col2, …) VALUES (val1, val2, …), (val3, val4, …), (val5, val6…) ;*

Il est également possible d'injecter des données directement tirées d'une autre table.

*INSERT INTO table1(col1, col2, …) SELECT col1, col2, … FROM table2 WHERE [condition] ;*

## Structured Query Language (SQL)

Il existe deux manières de supprimer des données au sein d'une table (cette suppression se base sur une base conditionnelle WHERE)

*DELETE FROM table1 WHERE [condition] ;*

Il est possible de supprimer l'ensemble des données sans condition au niveau de la table.

*DELETE FROM table1 ;*
*TRUNCATE FROM table1 ;*

La requête DELETE supprime les lignes tandis que la requête TRUNCATE réinitialise également les colonnes possédant une option d'auto-incrémentation. (Attention, la requête TRUNCATE ne nécessite pas de validation avec commit)

# Structured Query Language (SQL)

En production, l'utilisation la plus courante de SQL consiste à lire des données contenues dans la base de données.

Cette lecture s'opère au travers de la commande SELECT, qui retournes les tuples dans un tableau de résultat. Cette commande peut sélectionner une ou plusieurs colonnes d'une table.

## Utilisation basique

*SELECT nom_colonne FROM nom_table*

Cette requête SQL va sélectionner le champ (nom_colonne) au sein de la table nom_table. Il est possible de sélectionner plusieurs colonnes en les séparant d'une virgule (,)

## Structured Query Language (SQL)

Une requête SELECT peuvent être alimentées pour mieux définir les données que l'on souhaite lire. Il est possible de :

- Joindre un autre tableau aux résultats
- Filtrer le résultat pour ne sélectionner que certains enregistrements
- Ordonner les résultats
- Grouper les résultats pour en sortir des statistiques

Une requête SELECT peut ainsi vite devenir assez longue :

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

# TP – MARIADB INSTALLATION

# TP – GESTION AVEC PHPMYADMIN

# FIN !

Merci pour votre participation !