BASES DE DONNEES RELATIONNELLES AVEC MYSQL OU MARIADB

CODE COLLIDERS

CHRISTINE GUNSENHEIMER

ROMAIN VIRMAUX

Version 1.0 – 22/02/2020

TABLE DES MATIERES

Sites utiles et intéressants :

https://sql.sh/

https://sqlbolt.com/ (en)

https://fr.wikibooks.org/wiki/MySQL

https://www.w3resource.com/mysql/mysql-tutorials.php (en)

LIEN AVEC LE REFERENTIEL

Activités type n°2 : Développer la partie backend d'une application web ou web mobile en intégrant les recommandations de sécurité.

Compétence n°5 : Créer une base de données (en annexe de ce document).

QU'EST CE QU'UNE BASE DE DONNEES ?

Une base de données est une collection organisée de données. Elle permet le stockage et la récupération de données brutes. Les données sont rangées de façon à pouvoir les retrouver facilement (et rapidement).

Il existe de nombreux types de base de données (hiérarchique, relationnel, objet, graph, ...), qui organisent et structurent les données selon différents modèles.

Exemples de cas d'utilisation d'une base de données :

- Enregistrement des informations en provenance d'un formulaire de contact sur un site Internet
- Enregistrement des informations d'un compte utilisateur

En savoir plus sur les types de bases de données :

https://www.digora.com/fr/blog/TOP-10-des-bases-de-donnees-partie-2

https://www.tutorialspoint.com/Types-of-databases

QU'EST-CE QU'UNE BASE DE DONNEE RELATIONELLE ?

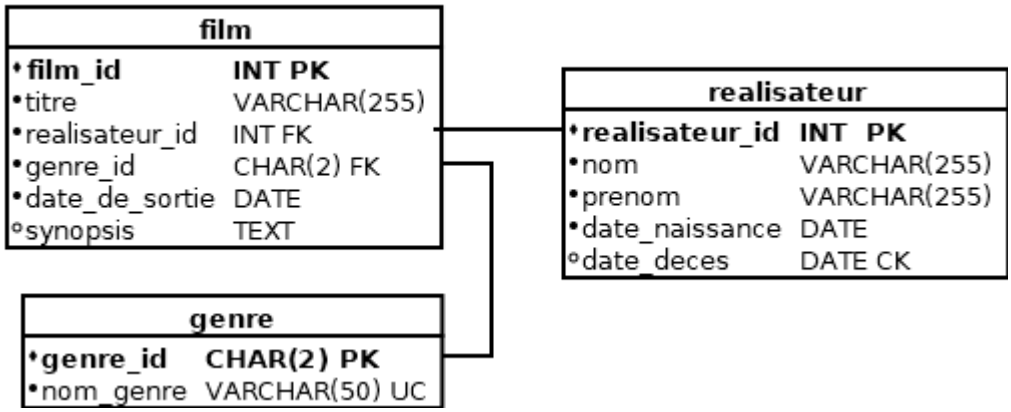
Une base de données relationnelles est une base de données ou les données sont organisés dans des tableaux à deux dimensions.

On appelle ces tableaux des tables. Une base de données contient plusieurs tables. Chaque table est constituée de colonnes.

Schéma physique de la table utilisateur

```
MariaDB [super_db]> SELECT * FROM utilisateur;
+----------------+--------+-----+-----------+-----------------------------+
| id_utilisateur | pseudo | nom | prenom    | email                       |
+----------------+--------+-----+-----------+-----------------------------+
|              1 | niamor | V.  | Romain    | romain@codecolliders.com    |
|              2 | noodle | G.  | Christine | christine@codecolliders.com |
+----------------+--------+-----+-----------+-----------------------------+
2 rows in set (0.001 sec)
```

Affichage des données contenues dans la table « utilisateur »



Exemple de schéma physique de base de données relationnelle. Cette base est composée de 3 tables, et 2 relations y sont schématisées.

SGBD ET SGBDR

Le Système de Gestion de Base de Données (Relationnelle) est un logiciel qui permet aux utilisateurs de définir, créer, maintenir et contrôler l'accès à une base de données (relationnelle).

Le SGBD garanti la cohérence des données selon un modèle normalisé.

- Toutes les colonnes contiennent des valeurs simples (non multiples, non composées).

- Toutes les colonnes non clés dépendent de la clé primaire.

- Il n'y a pas de dépendance fonctionnelle entre deux colonnes non clé.

SQL - STRUCTURED QUERY LANGUAGE

Le SQL est un langage de requête structuré permettant de communiquer avec le SGBDR pour effectuer les actions suivantes :

- définition des données (DDL: data description language)

- requêtage des données (DQL: data query language)

- manipulation des données (DML: data manipulation language)

- contrôle des accès aux données (DCL: data control language)

Les bases du SQL

En SQL les commentaires peuvent s'écrire des façons suivantes :

```
-- Ceci est un commentaire sur une ligne (tout ce qui se trouve derrière est commenté)
# Ceci est également un commentaire sur une ligne (tout ce qui se trouve derrière est commenté)
/* Ceci est un commentaire sur une ou plusieurs lignes (tout ce qui se trouve à l'intérieur est commenté) */
```

Chaque requête SQL doit se terminer par ‘;’

Conventions de nommage (nom des bases, tables et colonnes) :

- Utiliser uniquement les caractères de l’alphabet anglais.
- Ne pas utiliser d’espaces et les remplacer par des _
- Eviter d’utiliser des nombres.
- Ne pas utiliser de caractères accentués.
- Utiliser de préférence des minuscules.
- Utiliser des noms compréhensibles et précis.

Installation de MariaDB en ligne de commande (dans le terminal)

Voici la commande permettant d’installer MariaDB sur Ubuntu 18.04 :

» apt install mariadb-server

Une fois l’installation effectuée il faut exécuter le script suivant et répondre aux questions afin de sécuriser l’installation :

» mysql_secure_installation

Pour se connecter au SGBD depuis un terminal, il suffit de taper la commande :

» mysql -u nom_utilisateur -p

DDL : Définition d’une base de données

Dans l’exemple ci-dessous, en orange et en majuscule sont ce qu’on appelle les « mot clés » du langage. Ils ont une signification, par exemple CREATE DATABASE veut dire « créer la base de données ». On écrit ensuite le nom de la base que l’on souhaite créer.

```
CREATE DATABASE nom_base_de_donnees;
```

Les noms des bases de données, des tables et des colonnes peuvent être encadrés du caractère ` (touche AltGr + 7). Il est important d’encadrer le nom des bases, tables et colonnes lorsque le nom peut être confondu avec un mot clé du langage par exemple si la base s’appelle « create ».

```sql
CREATE DATABASE `create`;
```

Il peut arriver que l'on créer une base de données qui porte le même nom qu'une base de données existante dans ce cas, MySQL retourne une erreur. Afin d'éviter cette erreur il est possible d'utiliser la requête suivante.

```sql
CREATE DATABASE IF NOT EXISTS nom_base_de_donnees;
```

Le mot clé USE permet de spécifier la base de données sur laquelle les requêtes suivantes vont s'exécuter.

```sql
USE nom_base_de_donnees;
```

SHOW DATABASES permet d'afficher la liste des bases des données existantes.

```sql
SHOW DATABASES;
```

DCL : Gestion des permissions

Création d'un utilisateur ayant accès au SGBD en local et définition de son mot de passe.

```sql
CREATE USER 'nom_utilisateur'@'localhost' IDENTIFIED BY
'mot_de_passe_sécurisé';
```

Définition du mot de passe pour un utilisateur.

```sql
SET PASSWORD FOR 'nom_utilisateur'@'localhost' =
PASSWORD('mot_de_passe_sécurisé');
```

Suppression d'un utilisateur.

```sql
DROP USER IF EXISTS 'nom_utilisateur'@'localhost';
```

Afficher les droits de tous les utilisateurs.

```sql
SHOW GRANTS;
```

Afficher les droits de l'utilisateur courant.

```sql
SHOW GRANTS FOR CURRENT_USER;
```

Donner tous les droits à un utilisateur sur une base de données (et spécifier son mot de passe).

```sql
GRANT ALL PRIVILEGES ON nom_base_de_donnees.* TO '
nom_utilisateur '@'localhost' IDENTIFIED BY
'mot_de_passe_sécurisé';
```

Donner les droits de sélection, insertion, modification et suppression à un utilisateur sur une base de données.

```
GRANT SELECT, INSERT, UPDATE, DELETE ON nom_base_de_donnees.* TO
' nom_utilisateur '@'localhost';
```

Retire les droits à un utilisateur sur une base de données.

```
REVOKE ALL PRIVILEGES ON nom_base_de_donnees.* FROM '
nom_utilisateur '@'localhost';
```

Export de bases de données en ligne de commande

Pour exporter une base de données dans un fichier au format SQL, exécutez la commande suivante dans un terminal :

» mysqldump nom_base_de_donnees > fichier.sql

En fonction des droits, vous pouvez également spécifier le nom d'utilisateur à utiliser pour l'export.

» mysqldump –u user –p nom_base_de_donnees > fichier.sql

Execution d'un fichier SQL en ligne de commande

Afin d'exécuter un fichier au format SQL en ligne de commande saisissez l'une des deux commandes ci-dessous (selon les droits dont vous disposez) :

» mysql nom_base_de_donnees < fichier.sql

» mysql –u user –p nom_base_de_donnees < fichier.sql

Typage des données

- ### Types numériques entiers

Type	Description	Min	Max	Mémoire occupée
BOOLEAN	associe la valeur 0 à false et la valeur 1 à true	0	1	1
TINYINT	nombre entier	-128	127	1
SMALLINT	nombre entier	-32768	32767	2
MEDIUMINT	nombre entier	-8388608	8388607	3
INT **- ou INTEGER**	nombre entier	-2147483648	2147483647	4
BIGINT	nombre entier	-9223372036854775808	9223372036854775807	8

- ### Types numériques décimaux

Type	Description	Valeur par défaut	Exemples
DECIMAL(p,e) **- ou NUMERIC(p,e)**	nombre à virgule où 'p' indique la 'précision' ou nombre de chiffres en tout (max 65) et 'e' l'échelle ou le	DECIMAL est équivalent à DECIMAL(10,0)	DECIMAL(5,3) permet de stocker les nombres de –99,999 à 99,999.

	nombre de chiffres après la virgule (max 30)		
FLOAT(o),	nombre à virgule stocké sur le nb d'octet 'o'	FLOAT est équivalent à FLOAT(4),	FLOAT(4) permet de stocker des nombres de -3.402823466E+38 à -1.175494351E-38 et 0 et de 1.175494351E-38 à 3.402823466E+38
DOUBLE, REAL	nombre à virgule stocké sur 8 octets en MySQL	DOUBLE et REAL sont équivalents à FLOAT(8) en MySQL	FLOAT(8) permet de stocker des nombres de -1.7976931348623157E+308 à -2.2250738585072014E-308 et 0 et de 2.2250738585072014E-308 à 1.7976931348623157E+308

- **Types temporels**

Nom	Description	Format	min	max
DATE	date	YYYY-MM-DD	'1000-01-01'	'9999-12-31'
TIME	heure	HH:MM:SS	'-838:59:59.99999'	'838:59:59.99999'
DATETIME	date et heure au format string	YYYY-MM-DD HH:MM:SS	'1000-01-01 00:00:00.00'	'9999-12-31 23:59:59.99'
YEAR	année	YYYY	1901	2155
TIMESTAMP	date et heure au format int (stocké sur 4octet)	YYYYMMDDHHMMSS	Attention, ce n'est pas un vrai timestamp (nb de secondes depuis le 1er janvier 1970, 0 h 0 m 0s)	

- **Types alphanumériques courts**

Type	Description	Valeur par défaut	Mémoire occupée	Exemple
CHAR(n)	Chaîne de caractère courte (moins de 255 caractères) de 'n' caractères	CHAR est équivalent à CHAR(1)	'n' octets	CHAR(5) sert fréquemment à stocker les codes postaux
VARCHAR(n)	Chaîne de caractère courte (moins de 255 caractères) pouvant varier entre 0 et 'n' caratères		Nombre d'octets permettant de stocker le texte (max 'n' octets)	VARCHAR(255) sert fréquemment à stocker un nom

- **Types alphanumériques longs**

Type	Description	Longueur max	Mémoire occupée

TINYTEXT	Chaîne de caractères	255 caractères	Longueur de la chaîne + 1octet (max 2^8 octets)
TEXT	Chaîne de caractères	65 535 caractères	Longueur de la chaîne + 2octets (max 2^{16} octets)
MEDIUMTEXT	Chaîne de caractères	16 777 215 caractères	Longueur de la chaîne + 3octets (max 2^{24} octets)
LONGTEXT	Chaîne de caractères	4 294 967 295 caractères	Longueur de la chaîne + 4octets (max 2^{32} octets)

- **Types alphanumériques stockés sous forme binaire**

Type	Description	Mémoire occupée
BINARY(n)	Chaîne de caractère courte sous forme binaire de longueur 'n' bits. Equivalent à CHAR(n)	'n' octets
VARBINARY(n)	Chaîne de caractère courte sous forme binaire pouvant varier entre 0 et 'n' bits. Equivalent à VARCHAR(n)	Nombre d'octets permettant de stocker l'information (max 'n' octets)
TINYBLOB	Chaîne de caractère sous forme binaire de longueur max 255 bits. Equivalent à TINYTEXT	Longueur de la chaîne + 1octet (max 2^8 octets)
BLOB	Chaîne de caractère sous forme binaire de longueur max 65 535 bits. Equivalent à TEXT	Longueur de la chaîne + 2octets (max 2^{16} octets)
MEDIUMBLOB	Chaîne de caractère sous forme binaire de longueur max 16 777 215 bits. Equivalent à MEDIUMTEXT	Longueur de la chaîne + 3octets (max 2^{24} octets)
LONGBLOB	Chaîne de caractère sous forme binaire de longueur max 4 294 967 295 bits. Equivalent à LONGTEXT	Longueur de la chaîne + 4octets (max 2^{32} octets)

- **Types alphanumériques particuliers**

Type	Description	Longueur max	Valeurs non autorisées	Exemple
ENUM	Chaîne de caractères où des valeurs autorisées sont définies	65 535 valeurs possibles	MySQL stockera une chaîne vide '' dans le champ	ENUM('monsieur', 'madame', 'mademoiselle') peut servir à stocker la civilité
SET	Liste de chaînes de caractères où des valeurs autorisées sont définies	64 valeurs possible	MySQL stockera un 0 pour chaque valeur possible dans le champ	

DDL : Définition d'une table

Requête de création d'une table et de définition de l'ensemble de ses colonnes.

```
CREATE TABLE nom_table (
   nom_colonne_1        VARCHAR(255)       NULL,
   nom_colonne_2        INT                NOT NULL
);
```

Requête permettant la suppression d'une table.

```
DROP TABLE nom_table;
```

Requête permettant de supprimer tous les enregistrements d'une table.

```
TRUNCATE TABLE nom_table;
```

Requêtes retournant la définition des colonnes d'une table.

```
SHOW COLUMNS FROM nom_table;
```

```
DESCRIBE nom_table;
```

Requête permettant d'ajouter une colonne à une table.

```
ALTER TABLE nom_table ADD COLUMN nom_colonne VARCHAR(255);
```

Requête permettant la suppression d'une colonne.

```
ALTER TABLE nom_table DROP COLUMN nom_colonne;
```

Requête permettant de modifier une colonne.

```
ALTER TABLE nom_table MODIFY COLUMN nom_colonne VARCHAR(255);
```

Contraintes

- **NOT NULL**

La contrainte NOT NULL empêche une colonne d'accepter une valeur nulle. Si on tente d'insérer un enregistrement dont le champ a pour valeur nul, MySQL n'insérera pas l'enregistrement, génèrera une erreur et stoppera l'exécution du script.

Requête d'ajout de la contrainte NOT NULL sur une colonne :

```
ALTER TABLE nom_table MODIFY nom_colonne VARCHAR(255) NOT NULL;
```

Requête de suppression de la contrainte NOT NULL :

```
ALTER TABLE nom_table MODIFY nom_colonne VARCHAR(255) NULL;
```

Par défaut, les colonnes acceptent des valeurs nulles

- **UNIQUE**

La contrainte UNIQUE garantit que chaque valeur stockée dans un champs de la ou des colonne(s) possède une valeur différente. Si on tente d'insérer un enregistrement dont le champs a pour valeur une valeur déjà présente dans un autre champs de la colonne, MySQL n'insérera pas l'enregistrement, génèrera une erreur et stoppera l'exécution du script.

Requêtes d'ajout d'une contrainte UNIQUE sur une ou plusieurs colonnes :

```
ALTER TABLE nom_table ADD CONSTRAINT UC_nom_contrainte UNIQUE
(nom_colonne);

ALTER TABLE nom_table ADD CONSTRAINT UC_nom_contrainte UNIQUE
(nom_colonne_1, nom_colonne_2, nom_colonne_3);
```

Requête de suppression d'une contrainte UNIQUE :

```
ALTER TABLE nom_table DROP INDEX UC_nom_contrainte;
```

Une table peut contenir plusieurs colonnes avec des contraintes d'unicité. Les contraintes d'unicité peuvent également être appelée 'clés secondaires'.

- **PRIMARY KEY**

La contrainte PRIMARY KEY identifie de façon unique chaque enregistrement d'une table. Elle contient des valeurs uniques et non nulles. **Une table ne possède qu'une clé primaire au maximum**. La clé primaire peut être constituée d'une colonne (clé primaire simple) ou de plusieurs colonnes (clé primaire composée).

Requêtes d'ajout d'une clé primaire sur une ou plusieurs colonnes :

```
ALTER TABLE nom_table ADD CONSTRAINT PK_nom_table PRIMARY KEY
(nom_colonne);

ALTER TABLE nom_table ADD CONSTRAINT PK_nom_table PRIMARY KEY
(nom_colonne_1, nom_colonne_2, nom_colonne_3);
```

Requête de suppression d'une clé primaire :

```
ALTER TABLE nom_table DROP PRIMARY KEY;
```

- **FOREIGN KEY**

La contrainte FOREIGN KEY permet de référencer une ou plusieurs colonnes d'une table par rapport à une ou plusieurs colonnes d'une autre table. Elle identifie de façon unique un enregistrement (ligne) d'une autre table. Cette contrainte garantit qu'aucune action d'enregistrement, modification ou suppression ne détruise le lien entre deux tables.

Requêtes d'ajout d'une clé étrangère sur une ou plusieurs colonnes :

```sql
ALTER TABLE nom_table_1
    ADD CONSTRAINT FK_nom_table_1_nom_table_2 FOREIGN KEY
(nom_colonne_1) REFERENCES nom_table_2 (nom_colonne_2);

ALTER TABLE nom_table_1
    ADD CONSTRAINT FK_nom_table_1_nom_table_2
    FOREIGN KEY (nom_colonne_1, nom_colonne_2, ...)
    REFERENCES nom_table_2 (nom_colonne_3, nom_colonne_4, ...);
```

Requête de suppression d'une clé étrangère :

```sql
ALTER TABLE nom_table_1 DROP FOREIGN KEY
FK_nom_table_1_nom_table_2;
```

- **CHECK**

La contrainte CHECK s'assure que les valeurs stockées dans la colonne répondent à une condition particulière. Elle permet donc de restreindre les valeurs stockées dans la colonne. Si on tente d'insérer un enregistrement dont la valeur du champ ne satisfait pas la condition exprimée dans la contrainte CHECK, MySQL n'insérera pas l'enregistrement, génèrera une erreur et stoppera l'exécution du script. Cette contrainte peut se référer à des valeurs contenues dans une ou plusieurs colonnes

Requêtes d'ajout d'une contrainte CHECK à une ou plusieurs colonnes :

```sql
ALTER TABLE nom_table ADD CONSTRAINT CK_nom_contrainte CHECK
(colonne_1 > 0);

ALTER TABLE nom_table ADD CONSTRAINT CK_nom_contrainte CHECK
(colonne_1 > 0 AND colonne_2 = "valeur");
```

Requête de suppression de la contrainte CHECK :

```sql
ALTER TABLE nom_table DROP CHECK CK_nom_contrainte;
```

- **DEFAULT**

La contrainte DEFAULT définit une valeur par défaut pour une colonne. Cette valeur sera ajoutée au champ de l'enregistrement uniquement si aucune valeur n'est spécifiée.

Ajout de la contrainte DEFAULT sur une colonne :

```sql
ALTER TABLE nom_table ALTER nom_colonne SET DEFAULT "valeur_par
défaut";
```

Suppression de la contrainte DEFAULT d'une colonne :

```sql
ALTER TABLE nom_table ALTER nom_colonne DROP DEFAULT;
```

AUTO_INCREMENT

AUTO_INCREMENT n'est pas à proprement parler une contrainte. Il s'agit plutôt d'une propriété d'une colonne qui permet de générer la valeur qui sera stockée dans le champ de la colonne lors d'un enregistrement.

Il ne peut y avoir qu'une seule colonne possédant la propriété AUTO_INCREMENT par table La propriété AUTO_INCREMENT ne peut être mis en place que sur une colonne composant soit une clé primaire, soit une clé secondaire.

Exemple de définition de la propriété AUTO_INCREMENT lors de la création de la table :

La propriété AUTO_INCREMENT est mise en place lors de la création de la table.

```
CREATE TABLE nom_table (
    nom_colonne_1    VARCHAR(255)   NULL      AUTO_INCREMENT,
    nom_colonne_2    INT            NOT NULL
);
```

Exemple de requête d'ajout d'un AUTO_INCREMENT et clé primaire sur une colonne existante :

DDL : Définition des index

Un index permet d'accéder aux valeurs contenues dans les champs d'une colonne de manière plus efficace et rapide. Les index sont automatiquement créés pour les clés primaires et les clés secondaires mais doivent être manuellement mis en place pour les clés étrangères ou lorsqu'un grand nombre de requête est effectué sur une ou plusieurs colonnes.

Attention, chaque index prend de la place sur le disque et en mémoire.

Requête d'ajout d'un INDEX sur une colonne.

```
CREATE INDEX nom_index ON nom_table( nom_colonne ASC );
```

Requête d'ajout d'un INDEX sur plusieurs colonnes.

```
CREATE INDEX nom_index ON nom_table( nom_colonne_1,
nom_colonne_2, … DESC );
```

Requête d'ajout d'une contrainte UNIQUE sur une colonne.

```
CREATE UNIQUE INDEX nom_index ON nom_table (nom_colonne_1,
nom_colonne_2, … ASC);
```

Requête de suppression d'une contrainte INDEX sur une colonne.

```
DROP INDEX nom_index ON nom_table;
```

Requête permettant de reconstruire tous les index d'une table.

```
ALTER INDEX ALL ON nom_table REBUILD;
```

DML : Insertion d'enregistrements dans une table

Il existe deux manières d'insérer des données dans une table :

- La première consiste à spécifier le nom des colonnes et les valeurs correspondantes. Dans ce cas, il faut veiller au fait que toutes les colonnes non nulles soient mentionnées.

```
INSERT INTO nom_table ( nom_colonne_3 , nom_colonne_1 ) VALUES ( value_3 , value_1 );
```

- La seconde consiste à insérer une valeur pour toutes les colonnes. Dans ce cas, il faut veiller à bien respecter l'ordre des colonnes dans la table.

```
INSERT INTO nom_table VALUES ( valeur_1, valeur_2 , valeur_3, ... );
```

Lorsque vous souhaitez insérer des enregistrements dans une table ayant une colonne auto-incrémentée, préférez la première méthode en omettant de spécifier la colonne auto-incrémentée et sa valeur.

Attention à l'ordre dans lequel vous insérez vos données dans vos tables, en particulier si celles-ci comportent des clés étrangères. Si votre enregistrement fait référence à une valeur contenue dans un champs d'une autre table, il faut que cette valeur soit déjà enregistrée dans la seconde table.

DML : Modification des enregistrements contenus dans une table

Requête permettant de modifier un champ de tous les enregistrements déjà insérés dans une table.

```
UPDATE nom_table
    SET colonne_3 = valeur_3;
```

Requête permettant de modifier plusieurs champs de tous les enregistrements déjà insérés dans une table.

```
UPDATE nom_table
    SET colonne_3 = valeur_3, colonne_1 = valeur_1;
```

Requête permettant de modifier un champ des enregistrements déjà insérés dans une table avec une condition.

```
UPDATE nom_table
    SET colonne_1 = valeur_1
    WHERE condition;
```

Requête permettant de modifier plusieurs champs des enregistrements déjà insérés dans une table avec une condition.

```
UPDATE nom_table
    SET colonne_1 = valeur_1 , colonne_3 = valeur_3
    WHERE condition;
```

Attention à l'ordre dans lequel vous modifiez vos données dans vos tables, en particulier si celles-ci comportent des clés étrangères. Si votre enregistrement fait référence à une valeur contenue dans un champs d'une autre table, il faut que cette valeur existe déjà dans la seconde table.

DML : Suppression des enregistrements contenus dans une table

Requête permettant de supprimer tous les enregistrements d'une table.

```
DELETE FROM nom_table;
```

Requête permettant de supprimer tous les enregistrements d'une table satisfaisant une condition.

```
DELETE FROM nom_table
    WHERE condition;
```

Attention à l'ordre dans lequel vous supprimez les enregistrements, en particulier si vos tables comportent des clés étrangères. Si l'enregistrement que vous souhaitez supprimer est référencé par une valeur contenue dans un champs d'une autre table, il faut d'abord supprimer cette valeur de la seconde table.

DQL : Requêtage des enregistrements

Structure d'une requête

SELECT [DISTINCT] { * | colonne_1, colonne_2, ... }
FROM { nom_table }
[WHERE condition]
[GROUP BY colonne_1, colonne_2, …]
[ORDER BY colonne_1, colonne_2, … { ASC | DESC }]

[LIMIT nb_lignes] ;

1. SELECT

La clause SELECT permet de récupérer les enregistrements d'une base de données.

Dans sa forme la plus simple, SELECT se compose des colonnes que l'on souhaite récupérer et de la table dans laquelle ces colonnes sont contenues.

Requête permettant de récupérer tous les champs de tous les enregistrements d'une table (toutes les colonnes et toutes les lignes):

```
SELECT *
    FROM nom_table;
```

Requête permettant de ne récupérer que certains champs de tous les enregistrements d'une table (sélection de colonnes et toutes les lignes) :

```
SELECT nom_colonne_2 , nom_colonne_4
    FROM nom_table;
```

2. WHERE

La clause WHERE permet de filtrer les enregistrements retournés par la requête SELECT.

```
SELECT *
    FROM nom_table
    WHERE conditions;
```

- **Opérateurs de comparaison SQL**

Opérateur	Description	Exemple
=	Égal à	SELECT * FROM medecin WHERE nom = 'DUPONT' AND prenom = 'Jean'; SELECT id_vhs, id_etat FROM vhs WHERE id_film = 3;
<> ou !=	Pas égal à	SELECT DISTINCT nom FROM client WHERE nom <> 'Durand'; SELECT * FROM vhs WHERE id_etat != 1;
>	Supérieur à	SELECT id_film, titre FROM film WHERE annee_parution > '2005';
>=	Supérieur ou égal à	SELECT id_fleur FROM fournisseur_fleur WHERE stock >= 50;
<	Inférieur à	SELECT id_fleur FROM fleur WHERE prix < 45,00;

<=	Inférieur ou égal à	SELECT * FROM rendez-vous WHERE date_heure <= '2001-12-31 23:59:59';
IN (val1, val2, …)	Égal à l'une des valeurs de la liste	SELECT * FROM client WHERE code_postal IN (72000, 72100);
NOT IN (val1, val2, …)	Pas égal à l'une des valeurs de la liste	SELECT id_couleur FROM couleur WHERE libelle NOT IN ('rose', 'blanche', 'rouge');
BETWEEN 'val1' AND 'val2'	Compris dans un intervalle	SELECT id_variete FROM variete WHERE libelle BETWEEN 'rose' AND 'tulipe';
NOT BETWEEN 'val1' AND 'val2'	Pas compris dans l'intervalle	SELECT * FROM emprunt WHERE date_emprunt NOT BETWEEN '2019-01-01' AND '2019-12-31';
LIKE	Recherche d'une valeur selon un modèle	SELECT * FROM patient WHERE numero_secu LIKE '1%'; SELECT * FROM patient WHERE prenom LIKE '%ine'
NOT LIKE	Recherche d'une valeur ne correspondant pas à un modèle	SELECT * FROM film WHERE titre NOT LIKE '%jedi%'; SELECT * FROM film WHERE id_realisateur NOT LIKE '1%%2';
IS NULL	Le champ contient une valeur NULL	SELECT * FROM emprunt WHERE date_restitution IS NULL;
IS NOT NULL	Le champ contient n'importe quelle valeur tant que ce n'est pas NULL	SELECT * FROM patient WHERE telephone IS NOT NULL;

- **Opérateurs logiques SQL**

Opérateur	**Description**	**Exemple**
AND	Et (opérateur de conjonction)	SELECT * FROM medecin WHERE nom = 'DUPONT' AND prenom = 'Jean';
OR	Ou (opérateur de disjonction)	SELECT * FROM medecin WHERE nom = 'DUPONT' OR nom = 'DURAND';
NOT	Non (opérateur de négation)	SELECT id_couleur FROM couleur WHERE libelle NOT IN ('rose', 'blanche', 'rouge');

Attention lorsque vous combinez des opérateurs logiques.

Ex: les exemples ci-dessous ne retournerons pas les mêmes enregistrements :

SELECT * FROM film WHERE titre LIKE 'Une%' AND (annee_parution = '2019' OR id_realisateur = 5);

SELECT * FROM film WHERE (titre LIKE ‘Une%’ AND annee_parution = ’2019’) OR id_realisateur = 5;

SELECT * FROM film WHERE annee_parution = ’2019’ AND (titre LIKE ‘Une%’ OR id_realisateur = 5);

3. ORDER BY

La clause ORDER BY permet de trier les enregistrements retournés par la requête SELECT.

```
SELECT *
    FROM nom_table
    WHERE nom_colonne_2 > 5
    ORDER BY nom_colonne_3;
```

Par défaut, le tri des enregistrements s’effectue selon un ordre croissant. Toutefois, on peut spécifier l’ordre selon lequel on souhaite trier les enregistrements.

```
SELECT *
    FROM nom_table
    ORDER BY nom_colonne_3 ASC;
```

```
SELECT nom_colonne_2, nom_colonne_3
    FROM nom_table
    WHERE nom_colonne_4 > '2000-01-01 00:00:00'
    ORDER BY nom_colonne_3 DESC;
```

On peut également trier les enregistrements selon les valeurs contenues dans plusieurs champs. Dans le cas suivant, la requête retourne des enregistrements triés par ordre croissant sur la colonne 3 puis par ordre décroissant sur la colonne 2 puis par ordre décroissant sur la colonne 1.

```
SELECT *
    FROM nom_table
    ORDER BY nom_colonne_3 ASC, nom_colonne_2 DESC, nom_colonne_1 DESC;
```

4. LIMIT

L’utilisation de la commande SELECT peut potentiellement retourner une très grande quantité d’enregistrements. Si l’on souhaite récupérer uniquement une partie des enregistrements, on ajoute la clause LIMIT nb_de_lignes_à_retourner à la fin de la requête SQL.

Requête permettant de récupérer les 10 premiers enregistrements d’une table :

```
SELECT *
    FROM nom_table
    LIMIT 10;
```

Requête permettant de récupérer les 20 enregistrements suivants (de 11 à 30) d'une table :

```
SELECT *
    FROM nom_table
    LIMIT 10, 20;
```

5. GROUP BY

La clause GROUP BY permet de grouper des enregistrements par valeur contenue dans les champs d'une colonne

• Fonctions d'agrégation SQL

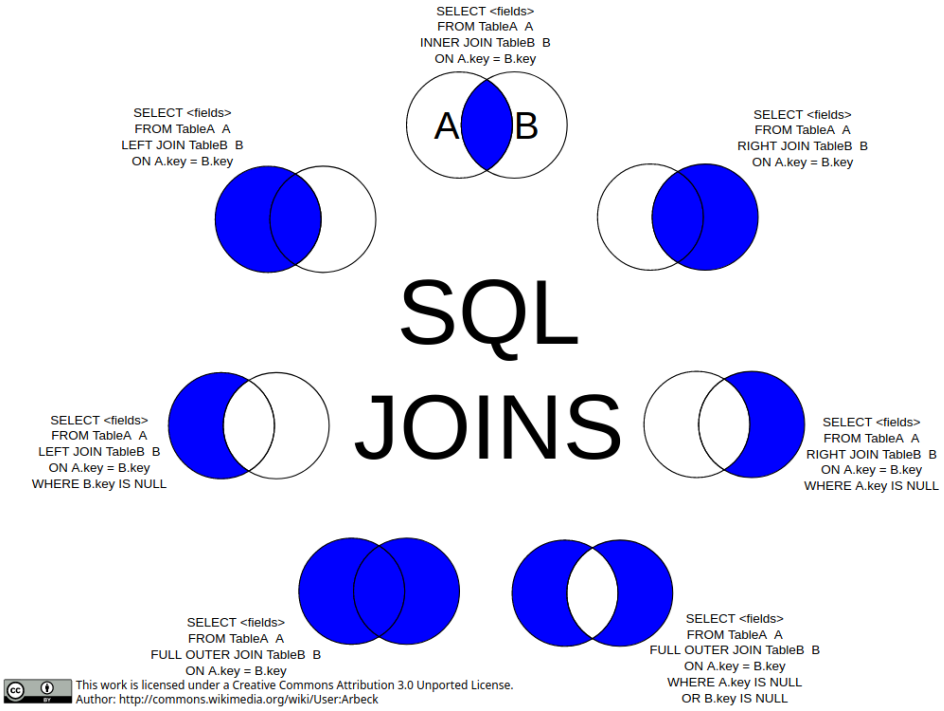
SQL contient nativement des fonctions d'agrégation qui permettent de réaliser opérations sur les enregistrements d'une table.

Fonction	Description	Exemple
MIN(colonne)	Valeur minimum d'une colonne sur un ensemble d'enregistrements	SELECT MIN(nom_colonne) FROM nom_table; SELECT colonne_1, MIN(colonne_2) FROM nom_table GROUP BY colonne_1;
MAX(colonne)	Valeur maximum d'une colonne sur un ensemble d'enregistrements	SELECT MAX(nom_colonne) FROM nom_table; SELECT colonne_1, MAX(colonne_2) FROM nom_table GROUP BY colonne_1;
SUM(colonne)	Somme sur une ensemble d'enregistrements	SELECT SUM(nom_colonne) FROM nom_table; SELECT colonne_1, SUM(colonne_2) FROM nom_table GROUP BY colonne_1
AVG(colonne)	Moyenne sur un ensemble d'enregistrements	SELECT AVG(nom_colonne) FROM nom_table; SELECT colonne_1, AVG(colonne_2) FROM nom_table

		GROUP BY colonne_1;
COUNT(colonne)	Compte le nombre d'enregistrements	SELECT COUNT (*) FROM nom_table; SELECT colonne_1, COUNT(colonne_2) FROM nom_table GROUP BY colonne_1;

6. Jointures

Les jointures en SQL permettent d'associer plusieurs tables dans une même requête. Cela permet d'obtenir des résultats qui combinent les données de plusieurs tables de manière efficace.



7. Alias

Les alias en SQL permettent de rendre plus compréhensible le nom d'une colonne ou d'une table retournée.

```
SELECT COUNT(nom_colonne_1) AS nb_enregistrement
    FROM nom_table;
```

Les alias permettent également raccourcir et simplifier les noms des tables lors de jointures.

```sql
SELECT *
    FROM nom_premiere_table AS t1
    INNER JOIN nom_deuxieme_table AS t2
    ON t1.nom_colonne_2 = t2.autre_nom_colonne ;
```

8. DISTINCT

L'utilisation de la commande SELECT peut potentiellement retourner des doublons. Si l'on souhaite éviter les répétitions, on ajoute le mot clé DISTINCT juste après SELECT. L'utilisation de la commande DISTINCT est très pratique pour éviter les résultats en doubles mais peut affecter les performances.

Requête permettant de récupérer tous les champs de tous les enregistrements uniques d'une table (toutes les colonnes et toutes les lignes uniques) :

```sql
SELECT DISTINCT *
    FROM nom_table;
```

Requête permettant de ne récupérer que certains champs des enregistrements d'une table puis que les enregistrements uniques dans cette sélection (sélection de colonnes et toutes les lignes uniques de cette sélection) :

```sql
SELECT DISTINCT nom_colonne_2 , nom_colonne_3
    FROM nom_table;
```