UTOPIOS®
TRANSFORMATION DIGITALE INTELLIGENTE

Formation SQL

Utopios® Tous droits réservés

Introduction

Ce module est centré sur le **langage SQL** (Structured Query Language)

Il s'agit d'un **langage de requêtes**.

Il est principalement utilisé pour la manipulation et la gestion de données.

Son rôle est **l'interaction** avec les **bases de données**, et plus spécifiquement les **bases de données relationnelles**.

Utopios® Tous droits réservés

UTOPIOS®
TRANSFORMATION DIGITALE INTELLIGENTE

Qu'est ce qu'une donnée ?

Utopios® Tous droits réservés

Les données

En informatique, les **données** (ou ***data*** en anglais) sont des **représentations d'informations**.

Nous pouvons les **stocker**, les **traiter** et les **manipuler**.

Elles peuvent prendre des formes ou des **types** variés.

Exemple: données binaires, numériques, textuelles…

Utopios® Tous droits réservés

Les données (data)

Je m'appelle Jean Dupont, je suis garagiste et j'ai 25ans

Je m'appelle Marie Dupuis, je suis chauffagiste et j'ai 33ans

Je m'appelle Eric François, je suis contrôleur et j'ai 51ans

Utopios® Tous droits réservés

Les données (data)

Je m'appelle Jean Dupont, je suis garagiste et j'ai 25ans

Je m'appelle Marie Dupuis, je suis chauffagiste et j'ai 33ans

Je m'appelle Eric François, je suis contrôleur et j'ai 51ans

Utopios® Tous droits réservés

Les données

Pour éviter que les données soient isolées, surtout si elles deviennent nombreuses, il devient nécessaire de les regrouper à l'intérieur d'ensembles appelés **Bases de données (BDD)** ou ***Database (DB)*** en anglais.

Utopios® Tous droits réservés

UTOPIOS®
TRANSFORMATION DIGITALE INTELLIGENTE

Bases de Données

Utopios® Tous droits réservés

Les Bases de données

Une base de données (ou BDD) est une **collection organisée et structurée** de données interdépendantes.

Elles peuvent s'organiser de manières différentes, appelées **modèles de base de données**. De nos jours, les deux modèles dominants sont:

Modèle relationnel (SQL)

Modèle non-relationnel (NoSQL)

Utopios® Tous droits réservés

Le modèle non-relationnel : NoSQL (NotOnlySQL)

Il peut se présenter sous diverses orientations:

- Bases de données orientées clés-valeurs
- Bases de données orientées graphe
- Bases de données orientées document

...

Mais il sera peu abordé durant ce module, qui traite du SQL à travers le modèle relationnel.

Utopios® Tous droits réservés

UTOPIOS®
TRANSFORMATION DIGITALE INTELLIGENTE

Bases de données relationnelles

Utopios® Tous droits réservés

Les bases de données relationnelles

Développées dès le début des années 70, les **bases de données relationnelles** (ou *relational databases*) sont l'un des modèles de base de données les plus utilisés.

Les BDDR sont basées sur la théorie des *relations entre données* et utilisent des **tables** composées de **lignes** et de **colonnes** pour les organiser.

Cela signifie qu'on va essayer de regrouper nos données à travers des caractéristiques ou des étiquettes **communes**.

Utopios® Tous droits réservés

Les données (data)

Je m'appelle Jean Dupont, je suis garagiste et j'ai 25ans

Je m'appelle Marie Dupuis, je suis chauffagiste et j'ai 33ans

Je m'appelle Eric François, je suis contrôleur et j'ai 51ans

Utopios® Tous droits réservés

Les données (data) + relations

Je m'appelle Jean Dupont, je suis garagiste et j'ai 25ans — Individu 1

Je m'appelle Marie Dupuis, je suis chauffagiste et j'ai 33ans — Individu 2

Je m'appelle Eric François, je suis contrôleur et j'ai 51ans — Individu 3

Utopios® Tous droits réservés

Les données (data) + relations

Je m'appelle Jean Dupont, je suis garagiste et j'ai 25ans

Je m'appelle Marie Dupuis, je suis chauffagiste et j'ai 33ans

Je m'appelle Eric François, je suis contrôleur et j'ai 51ans

Prénom Nom Métier Age

Utopios® Tous droits réservés

Les tables

Prénom	Nom	Métier	Age
Jean	Dupont	garagiste	25
Marie	Dupuis	chauffagiste	33
Eric	François	controleur	51

Utopios® Tous droits réservés

Les tables

Utopios® Tous droits réservés

Utopios® Tous droits réservés

Utopios® Tous droits réservés

Exercice 1

J'ai 4 individus avec certains de leurs attributs spécifiés.
Classez leurs données sous forme de table:

- Julie, femme, 24ans, électricienne, 4ans d'expérience
- Françoise, comptable, 40ans, 18ans d'expérience, femme
- Romain, 30ans, développeur, homme, expérience non spécifiée
- Manuel, 12 ans d'expérience, 39ans, homme, professeur

Utopios® Tous droits réservés

Exercice 1

Correction

A retenir: - Les doublons de données sont possibles
- Le champ d'une donnée peut être vide

Prénom	Age	Métier	Expérience	Genre
Julie	24	electricienne	4	F
Françoise	40	comptable	18	F
Romain	30	développeur	NULL	H
Manuel	39	professeur	12	H

Utopios® Tous droits réservés

Exercice 2

Voici un tas de données et d'entêtes isolées, regroupez les avec cohérence, -de façon relationnelle-, sous forme de table:

gros chien pigeon classe
chat petit
autruche nom éléphant
taille mammifere
herbivore cheval
oiseau moyen omnivore
baleine carnivore être humain régime alimentaire

Utopios® Tous droits réservés

Exercice 2

Correction

A retenir: – Les doublons de données sont possibles
– Le champ d'une donnée peut être vide

Nom	Taille	Régime alimentaire	Classe
chien	petit	carnivore	mammifère
chat	petit	carnivore	mammifère
baleine	gros	carnivore	mammifère
autruche	moyen	herbivore	oiseau
cheval	moyen	herbivore	mammifère
pigeon	petit	herbivore	oiseau
éléphant	gros	herbivore	mammifère
être humain	moyen	omnivore	mammifère

Utopios® Tous droits réservés

UTOPIOS®
TRANSFORMATION DIGITALE INTELLIGENTE

Les Systèmes de Gestion de Bases de Données (SGBD)

Utopios® Tous droits réservés

Les Systèmes de Gestion de Bases de Données

Aussi appelés **DataBase Management System (DBMS)**, ces **logiciels** nous facilitent le travail en permettant la **création** et la **définition** de la **structure** de nos BDD.

Mais ils permettent aussi surtout de faciliter le stockage, la manipulation et la gestion de données.

Ils offrent un panel d'outils étendu pour nous permettre une maintenance aisée de grande quantité de données.

Utopios® Tous droits réservés

Les Systèmes de Gestion de Bases de Données

- **Création et définition de structure**:
  Permet de définir tables, colonnes, clé primaires, contraintes et relations.
- **Stockage et gestion de données**
  Permet de stocker de façon organisée de grandes quantités de données, mais également de gérer leurs opérations d'insertion, de mise à jour et de suppression.

Utopios® Tous droits réservés

Les Systèmes de Gestion de Bases de Données

- **Langage de requête**
  Ils fournissent un langage pour interroger et manipuler les données de façon compréhensible et fluide
- **Gestion des transactions** (ACID)
- **Sécurité et autorisations**
  Ils permettent de définir des niveaux d'autorisation, de contrôler la confidentialité des données et de fournir des droits d'accès aux utilisateurs.

Utopios® Tous droits réservés

Les Systèmes de Gestion de Bases de Données

- **Sauvegarde et récupération**
  Ils offrent des fonctionnalités de sauvegarde et de restauration afin d'éviter la perte de données essentielles en cas de panne système.
- **Optimisation des performance**
  Ils utilisent des techniques d'optimisation telles que l'indexation, la mise en cache, la parallélisation des requêtes etc...

Utopios® Tous droits réservés

Les Systèmes de Gestion de Bases de Données

Exemples de SGBD **relationnels** (SGBDR) populaires:

- MySQL/MariaDB
- PostgreSQL

Utopios® Tous droits réservés

Les Systèmes de Gestion de Bases de Données

Exemples de SGBD **relationnels** (SGBDR) populaires:

- Oracle Database

ORACLE®

- Microsoft SQL Server

Microsoft® SQL Server®

- IBM DB2

IBM DB2

- SQLite

SQLite

Utopios® Tous droits réservés

Les Systèmes de Gestion de Bases de Données

Exemples de SGBD **non-relationnels** (SGBDNR) populaires :

- MongoDB

- Redis

Utopios® Tous droits réservés

UTOPIOS®
TRANSFORMATION DIGITALE INTELLIGENTE

Le langage SQL

Utopios® Tous droits réservés

Le langage SQL

Le langage SQL *(Structured Query Language)* est le langage de **requêtes** qui va nous permettre d'**interagir** avec notre base de données via le système de gestion de base de données.

Conçu au début des années 70 par IBM, il a été normé par l'American National Standards Institute (ANSI) **en 1986** puis mondialement par l'International Organization for Standardization (ISO) **en 1989**.

Il reçoit encore fréquemment des mises à jour (sa dernière date de 2023) et est le langage de requêtes le plus utilisé au monde.

Utopios® Tous droits réservés

Le langage SQL

Quelques mises à jours majeures:

- **SQL-92** *(1992)*
  Ajout des requêtes récursives et des déclencheurs
- **SQL:1999** *(1999)*
  Support des objets et des procédures stockées
- **SQL:2011** *(1992)*
  Ajout des tables temporaires et des opérations de fusion.
- **SQL:2023** *(2023)*
  Support des fichiers JSON

Utopios® Tous droits réservés

Les requêtes SQL

Les extensions de langage

Le choix de SGBDR n'est pas anodin pour l'apprentissage du SQL, car de nombreux SGBDR ont leur propre **extension du langage** SQL qui apportent des **fonctionnalités supplémentaires** et peuvent parfois modifier la syntaxe.

Attention donc, certaines syntaxes de requêtes qui sont valables dans un SGBDR auront des syntaxes différentes dans d'autres

Cependant, la logique derrière ces différentes syntaxes reste la plupart du temps très similaire.

Utopios® Tous droits réservés

Exemples d'extensions de langages

- **PL/SQL** *(Procedural Language)*
  C'est l'extension créée par Oracle. Elle ajoute des fonctionnalités de programmation telles que les boucles, les fonctions…
- **PL/pgSQL**
  C'est l'extension utilisée par le SGBD PostgreSQL. Sa logique est proche du PL/SQL.
- **T-SQL** *(Transact-SQL)*
  L'extension spécifique de Microsoft SQL Server et Sybase.

Utopios® Tous droits réservés

Les sous-langages

Le SQL est un langage très étendu qui s'est vu augmenté de nombreuses fonctionnalités au fil des années. Il permet de faire diverses manipulations avec les bases de données relationnelles.

Par souci de clarté, il est souvent découpé en 5 grandes sous-catégories de **fonctionnalités spécifiques**, elles sont appelées les **sous-langages** SQL.

Utopios® Tous droits réservés

Les sous-langages

Utopios® Tous droits réservés

Définition des sous-langages

(Les sous-langages seront étudiés en détail dans leurs sections respectives)

- **Data Query Language (DQL):** Le langage de requête de données est utilisé pour interroger une BDD et en lire ses données.
- **Data Manipulation Language (DML):** Le langage de manipulation de données est utilisé pour modifier, ajouter ou supprimer des données au sein d'une base de données.
- **Data Definition Language (DDL):** Le langage de définition de données est utilisé pour définir la structure des objets de la base de données

Utopios® Tous droits réservés

Définition des sous-langages

(Les sous-langages seront étudiés en détail dans leurs sections respectives)

- **Data Control Language (DCL):** Le langage de contrôle de données est utilisé pour gérer les autorisations et les privilèges d'accès à la base de données.
- **Transaction Control Language (TCL):** Le langage de contrôle des transactions est utilisé pour gérer spécifiquement les transactions dans une base de données.

Utopios® Tous droits réservés

UTOPIOS®
TRANSFORMATION DIGITALE INTELLIGENTE

Introduction à MySQL

Utopios® Tous droits réservés

Introduction à MySQL

Pour ce module, notre choix va se porter sur l'un des SGBDR les plus utilisés: **MySQL**. Il présente de nombreux atouts:

- **Cross-platform** : Compatible avec la plupart des systèmes d'exploitation: Windows, MacOS, Linux...
- **Populaire** : MySQL dispose d'une vaste communauté d'utilisateurs qui fournissent un support, partagent des connaissances et contribuent à son amélioration continue.
- **Gratuit** : MySQL est un logiciel open source, il est disponible gratuitement sous la licence publique générale GNU (GPL).

Utopios® Tous droits réservés

Choix de l'interface

Pour travailler sur nos bases de données depuis notre ordinateur, plusieurs possibilités d'interfaces s'offrent à nous, les deux plus accessibles étant:

- **CLI (Command Line Interface) :** l'interface en ligne de commandes est accessible depuis n'importe quel ordinateur via le terminal qui lui est intégré. Elle est plus flexible mais plus difficile à appréhender.
- **GUI (Graphical User Interface) :** l'interface graphique est plus accessible pour les débutants et les non-développeurs, mais la surcouche graphique engendre une consommation de ressources supérieure

Utopios® Tous droits réservés

Choix de l'interface graphique

Pour le moment, nous allons travailler depuis un GUI. MySQL est compatible avec plusieurs logiciels, les plus populaires étant:

- MySQL Workbench

- Dbeaver

- phpMyAdmin

- Navicat

Utopios® Tous droits réservés

Choix de l'interface graphique : MySQL Workbench

Pour ce cours, nous allons choisir **MySQL Workbench**. C'est l'outil officiel fourni par **Oracle**, le propriétaire de MySQL.

Il fournit donc une intégration complète de MySQL et est régulièrement mis à jour pour être compatible avec les dernières versions.

Il offre également un support multi-plateformes.

Utopios® Tous droits réservés

Installation de MySQL et MySQL WorkBench

Pour Windows:

1) Se rendre sur le site mysql.com et télécharger "MySQL Installer for Windows":
https://dev.mysql.com/downloads/
2) Ouvrir le fichier "mysql-installer-community" téléchargé
3) Choisir l'installation "Developer Default"
4) Choisir la config type "Development Computer" pour la configuration du serveur

Utopios® Tous droits réservés

Installation de MySQL et MySQL WorkBench

Pour Windows:

5) Choisir l'authentification forte (strong password) comme méthode d'authentification
6) Créer un password pour notre user "root"
7) Accepter les choix par défaut pour le reste de l'installation
8) Si l'installation s'est bien déroulée, MySQL Shell et MySQL Workbench devrait s'ouvrir

Utopios® Tous droits réservés

UTOPIOS®
TRANSFORMATION DIGITALE INTELLIGENTE

Le DQL (Data Query Language)

Utopios® Tous droits réservés

Introduction

Ce module est centré sur le **DQL** *(Data Query Language)* ou **langage de requête de données**
Il s'agit d'un **sous-langage** du langage SQL. Il est, comme son parent, un **langage de requête**.

Son unique rôle est la **récupération de données** au sein d'une base. Il permet d'**interroger une BDD** pour en récupérer **les informations** mais ne permet ni la modification, ni la suppression, ni l'ajout de données. Ces fonctions seront apportées par d'autres sous-langages.

Utopios® Tous droits réservés

Avant-Propos

Cette section traite du sous-langage DQL, qui ne nous permet pas d'injecter ou de supprimer nous-mêmes nos données.
Une base de données est donc fournie avec le cours pour permettre la manipulation du langage, elle se nomme **dql_bdd.sql**
Son implémentation dépendra de l'interface utilisée.

Si vous avez des connaissances sur d'autres sous-langages du SQL, n'hésitez pas à suivre le cours avec vos propres bases de données.

Utopios® Tous droits réservés

Choisir sa base de données

Avant d'envoyer des requêtes, il me faut choisir dans quelle base de données je souhaite travailler, nos deux premières requêtes seront:

1. Pour afficher la liste des BDD disponibles:

   ```
   SHOW DATABASES;
   ```

2. Pour utiliser celle de mon choix:

   ```
   USE nom_de_la_bdd;
   ```

Utopios® Tous droits réservés

Voir les tables présentes dans la BDD

Une fois que je suis dans la base de données de mon choix, je peux consulter la liste des tables présentes:

- Pour MySQL/MariaDB:

```
SHOW TABLES;
```

Utopios® Tous droits réservés

UTOPIOS®
TRANSFORMATION DIGITALE INTELLIGENTE

SELECT... FROM...

Utopios® Tous droits réservés

SELECT

En SQL, la commande **SELECT** est utilisée pour **récupérer des données** spécifiques à partir d'une base de données. C'est l'une des commandes les plus fondamentales et couramment utilisées mais aussi probablement l'instruction la plus complexe du langage SQL.

C'est également la commande **centrale** du sous-langage DQL, elle nous sera indispensable dans la quasi-totalité de nos requêtes.

Elle permet de récupérer **une, plusieurs ou toutes** les colonnes de notre table.

Utopios® Tous droits réservés

Structure de la commande

La commande se décompose en deux parties **SELECT** et **FROM** sous cette forme :

```
SELECT nom_de_la_colonne
FROM nom_de_la_table;
```

Utopios® Tous droits réservés

Sélectionner toutes les colonnes de ma table

Pour sélectionner toutes les colonnes d'une table, on utilise comme raccourci **l'étoile** `*` après notre SELECT :

```
SELECT *
FROM nom_de_la_table;
```

Utopios® Tous droits réservés

Sélectionner plusieurs colonnes de ma table

De la même manière, il est possible de récupérer plusieurs colonnes en une seule requête, il suffit de les séparer d'une virgule

```
SELECT colonne1, colonne2, colonne3
FROM nom_de_la_table;
```

Attention : On peut sélectionner plusieurs colonnes mais on évitera de sélectionner **plusieurs tables** lors d'une requête. (voir "Récupérer les colonnes de plusieurs tables")

Utopios® Tous droits réservés

Exercice : récupérer des données de notre table

Essayons maintenant nous-même de récupérer les données de la table **Users** de notre base de données.

1. Dans un premier temps, récupérez l'intégralité de la table **Users**
2. Modifiez la requête pour n'afficher que les colonnes **first_name**, **last_name** et **job**

Utopios® Tous droits réservés

Récupérer les colonnes de plusieurs tables

Si notre BDD comporte plusieurs tables, il est possible de récupérer les colonnes de ces différentes tables, mais il sera obligatoire de préciser le nom de la table avant chaque colonne

```
SELECT table1.colonne1, table2.colonne1, table2.colonne2
FROM table1, table2;
```

Attention : On nomme ce type de jointure une **jointure implicite**. C'est une méthode dépréciée et peu lisible, on préférera l'utilisation de **jointures explicites** que nous verrons plus loin dans le cours

Utopios® Tous droits réservés

UTOPIOS®
TRANSFORMATION DIGITALE INTELLIGENTE

SELECT WHERE

Utopios® Tous droits réservés

La clause WHERE

La clause **WHERE** est une commande qui vient se greffer à notre SELECT, elle a **une fonction de filtre**, elle permet de ne sélectionner que les données qui correspondent au(x) filtre(s) que nous aurons choisi(s).
Elle prend la forme d'une **expression conditionnelle.**

Note: Lorsque la clause WHERE est omise, toutes les lignes sont affichées (équivalent à WHERE TRUE).

Utopios® Tous droits réservés

Structure de la commande

La clause **WHERE** vient se situer directement derrière la clause **FROM**:

```
SELECT nom_de_la_colonne
FROM nom_de_la_table
WHERE instruction_conditionnelle;
```

Pour indiquer la condition que je souhaite, je vais utiliser les **opérateurs de comparaison**

Utopios® Tous droits réservés

Les opérateurs de comparaison

Opérateur	Description	Exemple
=	Égalité	WHERE nom = prenom
>	Supérieur à	WHERE age > 18
<	Inférieur à	WHERE days < 20
>=	Supérieur ou égal	WHERE number >= 30
<=	Inférieur ou égal	WHERE date <= 1995
<> ou !=	Différent de	WHERE age != 21

Utopios® Tous droits réservés

Exemple

Dans ma table **Users**, je ne souhaite garder que les utilisateurs qui ont moins de 30ans et afficher leur prénom, leur nom et leur âge, j'enverrai la requête suivante:

```
SELECT first_name, last_name, age
FROM Users
WHERE age < 30;
```

Utopios® Tous droits réservés

Comparaison de texte

Il est également possible de comparer des données sous forme de texte, dans ce cas il faudra entourer le texte entre apostrophes `'`

```
SELECT first_name, last_name, age
FROM Users
WHERE name = 'Alice';
```

Note: Les guillemets fonctionnent également, mais par convention il est préférable d'utiliser les quotes (apostrophes)

Utopios® Tous droits réservés

Exercice : filtrer les données de notre table avec WHERE

Essayons maintenant nous-même de filtrer les données de la table **Users** de notre base de données.

1. Dans une première requête, récupérez tous les utilisateurs dont le métier n'est pas développeur
2. Dans une seconde requête, récupérez tous les utilisateurs dont le prénom est John.
3. Dans une dernière requête, récupérez tous les utilisateurs dont le salaire est supérieur ou égal à 3000.

Utopios® Tous droits réservés

CORRECTION : filtrer les données avec WHERE

```
SELECT first_name, last_name, job
FROM Users
WHERE job != 'Developer';

SELECT first_name, last_name
FROM Users
WHERE first_name = 'John';

SELECT first_name, last_name, salary
FROM Users
WHERE salary >= 3000
```

Utopios® Tous droits réservés

UTOPIOS®
TRANSFORMATION DIGITALE INTELLIGENTE

Les opérateurs AND, OR et NOT

Utopios® Tous droits réservés

Les opérateurs AND et OR

En plus des opérateurs de comparaison, notre instruction conditionnelle WHERE a la possibilité d'utiliser des **opérateurs logiques**: **AND, OR** et **NOT**.

Ils donnent la possibilité de filtrer en cumulant les conditions:

- L'opérateur **AND** sélectionnera les lignes qui remplissent **toutes les conditions**
- L'opérateur **OR** sélectionnera les lignes qui remplissent **au moins l'une des conditions**

Utopios® Tous droits réservés

Exemple : AND

Je souhaite récupérer toutes les lignes dont l'utilisateur s'appelle David **ET** dont le métier est docteur.

```
SELECT first_name, last_name, job
FROM Users
WHERE first_name = 'David' AND job = 'Doctor';
```

Les utilisateurs s'appelant David mais n'étant pas docteur ne seront pas sélectionnés. Tout comme les docteurs ne s'appelant pas David. Les deux conditions doivent être **VRAI**

Utopios® Tous droits réservés

Exemple : OR

Je souhaite récupérer toutes les lignes dont l'utilisateur s'appelle David **OU** dont le métier est docteur.

```
SELECT first_name, last_name, job
FROM Users
WHERE first_name = 'David' OR job = 'Doctor';
```

Les utilisateurs s'appelant David mais n'étant pas docteur seront sélectionnés. Tout comme les docteurs ne s'appelant pas David. Seule **L'UNE** des deux conditions doit être **VRAI** pour être récupérée.

Utopios® Tous droits réservés

Exercice : utiliser les opérateurs OR et AND

Dans notre table **Users**, essayez de filtrer de cette manière:

1. Dans un première requête, récupérez tous les utilisateurs dont l'âge est inférieur à 30ans ou supérieur et égal à 35ans.
2. Récupérez ensuite tous les utilisateurs dont le métier est professeur et le salaire est supérieur à 2600.

Utopios® Tous droits réservés

CORRECTION : Filtrer les données avec WHERE

```
SELECT first_name, last_name, age
FROM Users
WHERE  age < 30 OR age >= 35;

SELECT first_name, last_name, age, salary
FROM Users
WHERE  job = 'Teacher' AND salary > 2600;
```

Utopios® Tous droits réservés

L'opérateur NOT

L'opérateur **NOT** est un opérateur logique de négation qui est utilisé pour **inverser** une condition.
Il est généralement utilisé en combinaison avec les opérations de comparaison pour obtenir le **résultat inverse** de la condition.

Attention : Ne pas confondre avec "!=" ou "<>". NOT est utilisé pour **inverser une condition logique**, tandis que != est utilisé pour comparer deux valeurs et vérifier si elles sont différentes. Ils peuvent cependant être parfois utilisés à des fins similaires.

Utopios® Tous droits réservés

Exemple : NOT

Je souhaite récupérer toutes les lignes dont l'utilisateur n'habite pas à New York.

```
SELECT first_name, last_name, location
FROM Users
WHERE NOT location = 'New York';
```

Utopios® Tous droits réservés

Combinaison d'opérateurs logiques

Ces trois opérateurs logiques sont **cumulables** au sein d'une seule requête. On utilisera les parenthèses `()` pour en définir l'ordre des priorités. Prenons la requête suivante:

```
SELECT first_name, last_name, location, job
FROM Users
WHERE location = 'New York' AND (job = 'Teacher' OR job = 'Developer');
```

Cette requête sélectionnera tous les utilisateurs dont la localisation est New York et qui sont professeurs ou développeurs.

Utopios® Tous droits réservés

Exercice: récapitulatif

Créez une requête qui permet de récupérer toutes les personnes qui sont nées à New York, dont le salaire est compris entre 3000 et 3500 (compris) et qui ne sont ni docteur ni avocat.

- Toutes les conditions doivent tenir en une seule requête
- Les trois opérateurs logiques : AND, OR et NOT doivent être utilisés.

Utopios® Tous droits réservés

Correction: récapitulatif

```sql
SELECT
  *
FROM
  Users
WHERE
  birth_location = 'New York'
  AND (salary >= 3000 AND salary <= 3500)
  AND NOT (job = 'Doctor' OR job = 'Lawyer');
```

Utopios® Tous droits réservés

UTOPIOS®
TRANSFORMATION DIGITALE INTELLIGENTE

DISTINCT

Utopios® Tous droits réservés

La clause DISTINCT

La clause **DISTINCT** est utilisée pour spécifier que les résultats d'une requête doivent inclure uniquement les **valeurs uniques** d'une colonne. En d'autres termes, elle permet d'**éliminer les doublons dans les résultats** de la requête.
Comme la plupart des clauses que nous verrons par la suite, elle est compatible avec WHERE, voici sa **syntaxe**:

```
SELECT DISTINCT nom_de_la_colonne1, nom_de_la_colonne2
FROM nom_de_la_table
```

Utopios® Tous droits réservés

Exemple

Je cherche à récupérer tous les jobs uniques qui existent à la localisation New York:

```
SELECT DISTINCT job
FROM Users
WHERE location = 'New York';
```

Utopios® Tous droits réservés

UTOPIOS®
TRANSFORMATION DIGITALE INTELLIGENTE

IN et BETWEEN

Utopios® Tous droits réservés

IN et BETWEEN

En plus des opérateurs logiques OR, AND et NOT, nous pouvons affiner le filtre WHERE grâce aux opérateurs IN et BETWEEN:

- **IN** permet de filtrer les données dans une requête en utilisant des **valeurs précises**.
- **BETWEEN** permet de filtrer les données dans une requête en fonction de **plages de valeurs**.

Utopios® Tous droits réservés

IN

Par exemple, si je recherche tous les utilisateurs qui viennent **spécifiquement** de Londres et de Paris

```
SELECT first_name, last_name, location
FROM Users
WHERE location IN ('Paris', 'London');
```

Si vous êtes attentifs, vous constaterez que cette requête est similaire à:

```
SELECT first_name, last_name, location
FROM Users
WHERE location = 'Paris' OR location = 'London';
```

Utopios® Tous droits réservés

IN + NOT

L'opérateur IN est cumulable avec NOT également donc si je recherche tous les utilisateurs qui **NE** viennent **PAS** de Londres et de Paris:

```
SELECT first_name, last_name, location
FROM Users
WHERE location NOT IN ('Paris', 'London');
```

Cette requête est similaire à:

```
SELECT first_name, last_name, location
FROM Users
WHERE location != 'Paris' AND location != 'London';
```

Utopios® Tous droits réservés

BETWEEN

La clause **BETWEEN** est utilisée pour récupérer une plage précise de valeurs. Par exemple si je veux récupérer une plage d'utilisateurs dont l'âge est compris entre 30 et 35 ans:

```
SELECT first_name, last_name, age
FROM Users
WHERE age BETWEEN 30 AND 35;
```

NOTE: Elle est inclusive, c'est à dire que les valeurs minimales et maximales sont incluses dans la récupération des données (dans notre exemple, les utilisateurs de 30 et 35ans seront inclus)

Utopios® Tous droits réservés

BETWEEN + NOT

Comme **IN**, **BETWEEN** peut être cumulé avec **NOT** pour exclure toute une plage de valeurs. Par exemple si je veux récupérer une plage d'utilisateurs dont l'âge n'est pas compris entre 30 et 35 ans:

```
SELECT first_name, last_name, age
FROM Users
WHERE age NOT BETWEEN 30 AND 35;
```

NOTE: Dans la même logique, les valeurs minimales et maximales sont également exclues (dans notre exemple, 30 et 35 sont donc exclus)

Utopios® Tous droits réservés

Exercice: Manipulation de données dans la table "Users"

Dans notre table **Users**, en utilisant au moins pour une requête IN et pour une autre BETWEEN:

1. Sélectionnez tous les enregistrements où le métier est "Engineer"
2. Sélectionnez les prénoms et les noms de famille des utilisateurs nés à Londres, Paris ou Berlin
3. Sélectionnez les utilisateurs dont l'âge est compris entre 25 et 35 ans
4. Sélectionnez les utilisateurs qui sont à la fois des développeurs et dont le salaire est supérieur à 2500€

Utopios® Tous droits réservés

Correction :

```sql
SELECT * FROM Users
WHERE job = 'Engineer';

SELECT first_name, last_name FROM Users
WHERE birth_location IN ('London', 'Paris', 'Berlin');

SELECT * FROM Users
WHERE age BETWEEN 25 AND 35;

SELECT * FROM Users
WHERE job = 'Developer' AND salary > 2500;
```

Utopios® Tous droits réservés

UTOPIOS®
TRANSFORMATION DIGITALE INTELLIGENTE

ORDER BY

Utopios® Tous droits réservés

ORDER BY

La clause **ORDER BY** permet de classer nos données par ordre ascendant ou descendant en ciblant une colonne.

Si elle cible une colonne qui contient du texte, elle classera donc par **ordre alphabétique**, si elle contient des nombres, du plus petit au plus grand

Utopios® Tous droits réservés

Syntaxe

```
SELECT first_name, age
FROM users
ORDER BY age;
```

Ici, les gens seront classés par âge. Bien sûr, cette clause est cumulable avec **WHERE**:

```
SELECT first_name, age
FROM users
WHERE age < 50
ORDER BY age;
```

Utopios® Tous droits réservés

ASC, DESC

Par défaut, l'ordre de tri est croissant. Mais il est également possible de ranger par ordre décroissant grâce à la clause **DESC**:

```
SELECT *
FROM users
ORDER BY birth_location DESC;
```

On peut également préciser **ASC** quand on trie par ordre croissant, mais cela mène au résultat que de ne pas apporter de précision, on gagne cependant en lisibilité.

Utopios® Tous droits réservés

ORDER BY (avec plusieurs colonnes)

```
SELECT *
FROM users
ORDER BY last_name DESC, age ASC;
```

Il est tout à fait possible d'utiliser **ORDER BY** avec plusieurs colonnes. L'ordre à son importance.
Dans l'exemple ci-dessus, les utilisateurs seront classés par leur nom **de façon descendante** puis par leur âge **de façon ascendante** si leur nom est identique.

Utopios® Tous droits réservés

UTOPIOS®
TRANSFORMATION DIGITALE INTELLIGENTE

LIMIT/OFFSET

Utopios® Tous droits réservés

LIMIT

La clause **LIMIT** est utilisé pour limiter le nombre de lignes retournées par une requête SELECT.
Elle permet de **contrôler la quantité de données affichées** ou récupérées dans le résultat de la requête.

La clause LIMIT est souvent utilisée en conjonction avec l'ordre ORDER BY pour **trier les résultats avant de limiter le nombre de lignes.**

Utopios® Tous droits réservés

Syntaxe

La clause LIMIT vient se placer tout à la fin de nos filtres, **après** le ORDER BY:

```
SELECT column1, column2, ...
FROM table
WHERE conditions
ORDER BY column
LIMIT number;
```

Utopios® Tous droits réservés

Exemple

```
SELECT first_name, last_name, salary
FROM Users
ORDER BY salary DESC
LIMIT 5;
```

Avec cet exemple, je vais d'abord classer mes salaires du plus élevé au plus bas. Puis je ne vais garder que les 5 premiers résultats, cette requête me permet donc de récupérer **les 5 utilisateurs avec le plus gros salaire** de ma base de données

Utopios® Tous droits réservés

OFFSET

La clause LIMIT peut être enrichie d'un **OFFSET**. Sa fonction est de **décaler notre résultat** du nombre de lignes que l'on souhaite:

```
SELECT first_name, last_name, salary
FROM Users
ORDER BY salary DESC
LIMIT 5 OFFSET 3;
```

Si on reprend l'exemple précédent, mon résultat sera les **5 meilleurs salaires de ma table en excluant les 3 premiers**, donc, du 4ème au 8ème.

Utopios® Tous droits réservés

Exercice: Utilisation de ORDER BY, LIMIT et OFFSET

Toujours au sein de notre table Users, construisez les requêtes suivantes:

1. Sélectionnez les cinq utilisateurs les plus âgés de la table "Users", triés par ordre décroissant d'âge.
2. Affichez les enregistrements 6 à 10 triés par ordre alphabétique du prénom.
3. Sélectionnez les trois utilisateurs ayant les salaires les plus élevés de la table "Users", triés par ordre décroissant de salaire.

Utopios® Tous droits réservés

CORRECTION: Utilisation de ORDER BY, LIMIT et OFFSET

```sql
SELECT * FROM Users
ORDER BY age DESC
LIMIT 5;

SELECT * FROM Users
ORDER BY first_name
LIMIT 5 OFFSET 5;

SELECT * FROM Users
ORDER BY salary DESC
LIMIT 3;
```

Utopios® Tous droits réservés

UTOPIOS®
TRANSFORMATION DIGITALE INTELLIGENTE

Les fonctions d'agrégation

Utopios® Tous droits réservés

Les fonctions d'agrégation

Les **fonctions d'agrégation** en SQL nous permettent de réaliser des calculs sur les valeurs de notre base de données et d'en **retourner un résultat**. Elles sont nombreuses et dépendent du SGBD, nous verrons ici les plus utilisées:

MIN() - Retourne la valeur minimale d'une colonne.
MAX() - Retourne la valeur maximale d'une colonne.
COUNT() - Compte le nombre de lignes d'une colonne
SUM() - Calcule la somme des valeurs d'une colonne
AVG() - Calcule la valeur moyenne d'une colonne

Utopios® Tous droits réservés

MIN()/MAX()

Les fonctions MIN() et MAX() sont les fonctions d'agrégation les plus faciles à utiliser. Comme leur nom l'indique, elles retournent simplement la plus grande ou la plus petite valeur d'une colonne:

```
SELECT MIN(age)
FROM users;
SELECT MAX(age)
FROM users;
```

Les fonctions d'agrégation **s'appliquent sur les colonnes**, donc directement après le SELECT. Le filtre du WHERE s'applique **AVANT** l'agrégation

Utopios® Tous droits réservés

LES ALIAS

Comme nos fonctions d'agrégation nous retournent des résultats externes à notre BDD, il peut être bon de les renommer pour savoir à quoi ils correspondent.

Les **ALIAS** permettent de **RENOMMER** le nom d'un résultat. Ils fonctionnent grâce à la clause **AS**:

```
SELECT MIN(age) AS developer_min_age
FROM users
WHERE job = "Developer";
```

Ici j'ai renommé mon résultat en "developer_min_age"

Utopios® Tous droits réservés

Particularités

Les fonctions MIN() et MAX() fonctionnent également sur **les chaînes de caractères** (String), dans ce cas la valeur retournée est basée sur **l'ordre alphabétique**:

```
SELECT MAX(last_name) AS last_name
FROM users;
```

Ici, je récupérerai le nom de famille qui est le dernier par ordre alphabétique.

Utopios® Tous droits réservés

COUNT()

La fonction COUNT() est elle aussi assez simple d'utilisation, elle renvoie **le nombre de lignes** qui correspondent à notre critère.

```
SELECT COUNT(*) AS total_users
FROM users;
```

Ici, cette requête me permet de savoir le **nombre total d'utilisateurs** que comporte ma table.

Note: La fonction COUNT() ne compte pas les lignes dont la valeur est NULL, attention donc à votre choix de colonne.

Utopios® Tous droits réservés

SUM()

La fonction d'agrégation **SUM()** nous permet de calculer **la somme des différentes lignes d'une colonne**. Elle ne fonctionne donc qu'avec des nombres

```
SELECT SUM(salary) AS total_salary_without_devs
FROM users;
WHERE job != 'Developer';
```

Grâce à cette requête, je connais **le montant total des salaires** de tous mes utilisateurs en excluant celui des développeurs.

Utopios® Tous droits réservés

AVG()

La fonction d'agrégation **AVG()** fonctionne dans la même logique que SUM(). Mais elle retournera la moyenne de toutes les valeurs plutôt que la somme.

```
SELECT AVG(salary) AS average_salary_devs
FROM users;
WHERE job = 'Developer';
```

Cette requête me retournera la moyenne des salaires des développeurs.

Utopios® Tous droits réservés

Exercice: Utilisation des fonctions d'agrégation

Dans ma table Users, trouvez les requêtes suivantes **en utilisant à chaque fois des alias**:

1. Quel est le salaire minimum parmi tous les utilisateurs ?
2. Quel est l'âge maximum parmi les utilisateurs ayant le métier "Engineer" ?
3. Trouvez le salaire moyen des utilisateurs dont le métier est "Teacher".
4. Trouvez le montant total des salaires de tous les utilisateurs.

Utopios® Tous droits réservés

CORRECTION: Utilisation des fonctions d'agrégation

```sql
SELECT MIN(salary) AS MinSalary
FROM Users;

SELECT MAX(age) AS MaxAgeEngineer
FROM Users
WHERE job = 'Engineer';

SELECT AVG(salary) AS AvgSalaryTeacher
FROM Users
WHERE job = 'Teacher';

SELECT SUM(salary) AS TotalSalaries
FROM Users;
```

Utopios® Tous droits réservés

UTOPIOS®
TRANSFORMATION DIGITALE INTELLIGENTE

GROUP BY

Utopios® Tous droits réservés

GROUP BY

La clause GROUP BY en SQL est utilisée pour **regrouper les lignes de données en fonction des valeurs d'une ou plusieurs colonnes.**

Elle permet de créer des **groupes distincts de lignes** qui partagent des valeurs similaires dans les colonnes spécifiées.

La clause GROUP BY est **quasiment toujours** utilisée avec des **fonctions d'agrégation** (comme SUM(), AVG(), COUNT(), etc.) pour effectuer des calculs sur chaque groupe de données plutôt que sur l'ensemble complet de données.

Utopios® Tous droits réservés

```sql
SELECT birth_location, SUM(salary)
FROM Users
GROUP BY birth_location;
```

Cet exemple sert à calculer la somme des salaires pour chaque lieu de naissance distinct dans la table "Users".

- `SELECT birth_location, SUM(salary)`: Cela indique que nous voulons sélectionner la colonne "birth_location" (lieu de naissance) et la somme de la colonne "salary" (salaire).
- `GROUP BY birth_location`: Cela signifie que les lignes avec le même lieu de naissance seront regroupées ensemble.

Utopios® Tous droits réservés

Exemple

```
SELECT birth_location, AVG(salary)
FROM Users
GROUP BY birth_location
ORDER BY AVG(salary) DESC;
```

Cette requête sert à calculer la moyenne des salaires pour chaque lieu de naissance distinct dans la table "Users" et à les trier par ordre décroissant en fonction de la moyenne des salaires.

`ORDER BY AVG(salary) DESC`: Cette clause ordonne les résultats en fonction de la moyenne des salaires, triée de manière décroissante (DESC). Cela signifie que les lieux de naissance avec les moyennes de salaires les plus élevées seront affichés en premier.

Utopios® Tous droits réservés

UTOPIOS®
TRANSFORMATION DIGITALE INTELLIGENTE

HAVING

Utopios® Tous droits réservés

HAVING

La clause HAVING est plus simple, elle agit de la même façon qu'un WHERE mais au sein d'un groupement généré par un GROUP BY. Elle doit donc être située après celui-ci:

```
SELECT birth_location, AVG(salary) AS average_salary
FROM Users
GROUP BY birth_location
HAVING AVG(salary) > 3000;
```

Ici, j'applique mon filtre en disant que je ne veux que les groupes de lieux dont le salaire moyen est supérieur à 3000.

Utopios® Tous droits réservés

Remarque

L'exemple précédent est différent de:

```
SELECT birth_location, SUM(salary) AS total_salary
FROM Users
WHERE salary > 3000
GROUP BY birth_location;
```

En effet, ici, j'applique le filtre avant de grouper par lieux de naissance et avant la somme de mes salaires. Je dois donc bien réfléchir si je veux filtrer avant ou après et utiliser WHERE et HAVING en conséquence (les deux sont bien sûr cumulables)

Utopios® Tous droits réservés

UTOPIOS®
TRANSFORMATION DIGITALE INTELLIGENTE

LIKE

Utopios® Tous droits réservés

LIKE

Jusqu'à présent, nous avons filtré en utilisant des valeurs exactes, notamment grâce à nos opérateurs de comparaison. Mais il est également possible de filtrer en se basant sur des **patterns de texte**, c'est l'utilité de LIKE.

```
SELECT *
FROM Users
WHERE birth_location LIKE 'P%';
```

Cette requête cherchera tous les utilisateurs dont la ville de naissance commencent par la lettre P, c'est le rôle du `%`, il s'agit d'un **caractère générique**

Utopios® Tous droits réservés

Caractères génériques

En MySQL, vous pouvez utiliser deux caractères génériques avec l'opérateur LIKE pour effectuer des correspondances de motifs plus flexibles lors de la recherche de chaînes de caractères :

- `%` : Représente n'importe quelle séquence de caractères, y compris aucune séquence. Par exemple, si vous utilisez `LIKE 'a%'`, cela renverra toutes les valeurs qui commencent par la lettre 'a'.
- `_` : Représente exactement un caractère. Par exemple, si vous utilisez `LIKE '_o%'`, cela renverra toutes les valeurs où la deuxième lettre est 'o'.

Utopios® Tous droits réservés

Quelques exemples de caractères génériques

- `LIKE 'a%'` : Recherche de toutes les valeurs commençant par 'a'.
- `LIKE '%o'` : Recherche de toutes les valeurs se terminant par 'o'.
- `LIKE '%or%'` : Recherche de toutes les valeurs contenant 'or' n'importe où à l'intérieur du mot.
- `LIKE '_at%'` : Recherche de toutes les valeurs où la deuxième lettre est 'a' et la troisième lettre est 't'.
- `LIKE 'a%e'` : Recherche de toutes les valeurs commençant par 'a' et se terminant par 'e'.

Utopios® Tous droits réservés

Exercice : utilisation du LIKE

Dans notre table USERS, construisez les requêtes suivantes:

1. Sélectionnez les utilisateurs ayant un prénom qui commence par "D".
2. Trouvez les utilisateurs dont le nom de famille se termine par "son".
3. Identifiez les utilisateurs dont le prénom contient exactement 5 caractères.
4. Sélectionnez les utilisateurs ayant "Doctor" dans leur métier. (caractère générique obligatoire)

Utopios® Tous droits réservés

CORRECTION : utilisation du LIKE

```sql
SELECT *
FROM Users
WHERE first_name LIKE 'D%';

SELECT *
FROM Users
WHERE last_name LIKE '%son';

SELECT *
FROM Users
WHERE first_name LIKE '_____' ;

SELECT *
FROM Users
WHERE job LIKE '%Doctor%';
```

Utopios® Tous droits réservés

UTOPIOS®
TRANSFORMATION DIGITALE INTELLIGENTE

Sous-requêtes (Subqueries)

Utopios® Tous droits réservés

Sous-requêtes

En SQL, il est également possible d'**imbriquer** les requêtes les unes dans les autres pour gagner en précision. On nomme ce concept **la sous-requête**.

Il faut éviter d'entrer dans des niveaux de complexité trop hauts pour garder de la lisibilité.

Utopios® Tous droits réservés

Syntaxe

```
SELECT first_name, last_name, salary
FROM Users
WHERE salary > (SELECT AVG(salary) FROM Users);
```

Dans cet exemple, la sous-requête `(SELECT AVG(salary) FROM Users)` calcule la moyenne des salaires de tous les utilisateurs. La requête principale sélectionne ensuite les utilisateurs ayant un salaire supérieur à cette moyenne.

Utopios® Tous droits réservés

Autre exemple

```
SELECT first_name, last_name, salary
FROM Users
WHERE salary = (SELECT MAX(salary) FROM Users);
```

Dans cet exemple, la sous-requête `(SELECT MAX(salary) FROM Users)` calcule le salaire maximum parmi tous les utilisateurs. La requête principale sélectionne ensuite les utilisateurs ayant ce salaire maximum.

Utopios® Tous droits réservés

UTOPIOS®
TRANSFORMATION DIGITALE INTELLIGENTE

Les jointures

Utopios® Tous droits réservés

Introduction

Cette partie est centrée sur les **jointures** qui sont une fonctionnalité avancée du **langage DQL**.

Avant d'aborder le concept de jointure, il est important d'être à l'aise avec le langage DQL, et de savoir créer BDD et tables.

Utopios® Tous droits réservés

Définition

En DQL, nous avons appris à récupérer les données d'**une** table. Mais il est en fait beaucoup plus fréquent que nous ayons à **récupérer les données de plusieurs tables** en même temps au sein d'une seule requête.

Les jointures nous permettent cela, on les trouve principalement sous quatres formes: **INNER JOIN, RIGHT JOIN, LEFT JOIN et FULL JOIN**

Utopios® Tous droits réservés

History_Grades

student_id	student_name	history_grade
1	Alice	A
2	Jane	B
3	Julie	A

Math_Grades

student_id	student_name	math_grade
1	Alice	B
4	Roger	A
5	Kate	C

INNER JOIN

student_id	student_name	history_grade	student_id	student_name	math_grade
1	Alice	A	1	Alice	B

LEFT JOIN

student_id	student_name	history_grade	student_id	student_name	math_grade
1	Alice	A	1	Alice	B
2	Jane	B			
3	Julie	A			

FULL JOIN

student_id	student_name	history_grade	student_id	student_name	math_grade
1	Alice	A	1	Alice	B
2	Jane	B			
3	Julie	A			
			4	Roger	A
			5	Kate	C

RIGHT JOIN

student_id	student_name	history_grade	student_id	student_name	math_grade
1	Alice	A	1	Alice	B
			4	Roger	A
			5	Kate	C

Utopios® Tous droits réservés

Mise en situation

Imaginons que nous sommes dans le contexte d'un site d'abonnements à un service qui propose à ses clients **trois types d'abonnements**:

- L'abonnement STANDARD
- L'abonnement PREMIUM
- L'abonnement VIP

Et nous avons d'autres part **les clients**. Comment rattacher les clients à leurs abonnements ?

Utopios® Tous droits réservés

UTOPIOS®
TRANSFORMATION DIGITALE INTELLIGENTE

Lier les tables:
La clé étrangère (Foreign Key)

Utopios® Tous droits réservés

La clé étrangère

Avant de vouloir manipuler nos données, il est important de préparer nos tables à ce qu'elles communiquent entre elles. Ce moyen en SQL, c'est **la clé étrangère** (foreign key).
Elle se précise dès la construction de la table, il est donc préférable d'anticiper l'interaction entre les tables lorsque je construis une base de données.

Utopios® Tous droits réservés

Syntaxe

Pour avoir une jointure, il me faut d'abord deux tables, d'une part, **ma table Clients**:

```
CREATE TABLE Clients (
    id INT PRIMARY KEY AUTO_INCREMENT,
    first_name VARCHAR(100),
    last_name VARCHAR(100),
    city VARCHAR(100),
    age INT
  );
```

Utopios® Tous droits réservés

Syntaxe

D'autre part, **ma table Abonnements**, que je lie à ma table principale (ma table Clients ici):

```
CREATE TABLE Abonnements (
    client_id INT,
    abonnement_type VARCHAR(100),
    FOREIGN KEY (client_id)
    REFERENCES Clients(id)
);
```

Voilà, mes deux tables sont maintenant liées, c'est à dire que je ne pourrai plus insérer dans ma colonne client_id des valeurs qui n'existent pas déjà dans la colonne ID de ma table Client.

Utopios® Tous droits réservés

UTOPIOS®
TRANSFORMATION DIGITALE INTELLIGENTE

INNER JOIN

Utopios® Tous droits réservés

INNER JOIN

Il s'agit de ma première forme de jointure, en français on parle de **jointure intérieure**:

Son rôle est de connecter les lignes de deux tables (ou plus) en se basant sur une colonne qui les relie dans chacune.

Utopios® Tous droits réservés

Syntaxe

```
SELECT Clients.first_name, Clients.last_city, Abonnements.abonnement_type
FROM Clients
INNER JOIN Abonnements
ON Clients.id = Abonnements.client_id;
```

La clause `INNER JOIN` lie les deux tables Clients et Abonnements.
Le mot-clé `ON` est utilisé pour préciser la condition de jonction. Ici, c'est que l'id du client corresponde à celui du client_id de Abonnements.

Utopios® Tous droits réservés

UTOPIOS®
TRANSFORMATION DIGITALE INTELLIGENTE

Les alias de tables

Utopios® Tous droits réservés

Les alias de tables

Lorsqu'on en vient à manipuler plusieurs tables, il devient fastidieux de préciser à chaque fois l'entièreté de la table avant chaque colonne. Mais l'**alias de table** nous permet, comme son nom l'indique, de pouvoir renommer notre table, exactement de la même manière qu'on le faisait pour les colonnes.

```
SELECT c.first_name, c.last_city, a.abonnement_type
FROM Clients AS c
INNER JOIN Abonnements AS a
ON c.id = a.client_id;
```

Utopios® Tous droits réservés

Particularité : joindre plus de deux tables

```
SELECT
  c.name,
  p.product,
  p.amount
FROM
  Clients AS c
  JOIN Abonnements AS a ON c.id = a.customer_id
  JOIN Products AS p ON a.product_id = p.product_id;
```

Utopios® Tous droits réservés

UTOPIOS®
TRANSFORMATION DIGITALE INTELLIGENTE

LEFT JOIN/RIGHT JOIN

Utopios® Tous droits réservés

LEFT JOIN/RIGHT JOIN

Une jointure gauche (ou son inverse la jointure droite) va récupérer l'entièreté de la table de gauche et va venir y ajouter les colonnes de la table de droite qui lui sont liées:

On nomme left table: la table qui est mentionné la première dans la requête, la deuxième est la table de droite

Utopios® Tous droits réservés

Syntaxe

```
SELECT *
FROM Clients
LEFT JOIN Abonnements ON Clients.id = Abonnements.client_id;

SELECT *
FROM Clients
RIGHT JOIN Abonnements ON Clients.id = Abonnements.client_id;
```

Si aucune correspondance n'est trouvée dans la table de droite, les colonnes de la table de droite auront des valeurs NULL.

Utopios® Tous droits réservés

UTOPIOS®
TRANSFORMATION DIGITALE INTELLIGENTE

FULL JOIN

Utopios® Tous droits réservés

FULL JOIN

Enfin, la FULL JOIN est simplement la jointure totale de l'ensemble des deux tables

ATTENTION: La clause FULL JOIN n'est pas prise en charge dans toutes les bases de données, notamment MySQL.

Utopios® Tous droits réservés

FULL JOIN en MySQL

En MySQL, on utilisera une combinaison de LEFT JOIN et RIGHT JOIN pour simuler un FULL JOIN grâce à la clause **UNION**.

```
SELECT *
FROM Clients
LEFT JOIN Abonnements ON Clients.id = Abonnements.client_id

UNION

SELECT *
FROM Clients
RIGHT JOIN Abonnements ON Clients.id = Abonnements.client_id
WHERE Clients.id IS NULL;
```

Utopios® Tous droits réservés

Exercice: Découverte des jointures

Considérez deux tables, "Clients" et "Achats".

Création des tables :

- La table "Clients" a les colonnes suivantes : `id` (clé primaire), `nom`, `prenom`, `ville`.
- La table "Achats" a les colonnes suivantes : `client_id` (clé étrangère vers "Clients"), `produit`, `montant`.

Utopios® Tous droits réservés

Insertion des données :

- Insérez au moins cinq clients dans la table "Clients".
- Insérez au moins dix achats dans la table "Achats", en veillant à ce que certains clients aient effectué des achats et d'autres pas.

Utopios® Tous droits réservés

INNER JOIN :

- Sélectionnez les noms et prénoms des clients ainsi que les détails de leurs achats (si disponibles).

LEFT JOIN :

- Sélectionnez tous les clients et les détails de leurs achats s'ils ont effectué des achats, sinon affichez les colonnes des achats avec des valeurs NULL.

RIGHT JOIN :

- Sélectionnez tous les achats et les détails des clients correspondants, même s'il n'y a pas de correspondance.

Utopios® Tous droits réservés

FULL JOIN :

- Sélectionnez tous les clients et tous les achats, en affichant les détails des clients même s'ils n'ont pas effectué d'achats, et vice versa.

Utopios® Tous droits réservés

Correction :

a. INNER JOIN :

```
SELECT Clients.nom, Clients.prenom, Achats.produit, Achats.montant
FROM Clients
INNER JOIN Achats ON Clients.id = Achats.client_id;
```

b. LEFT JOIN :

```
SELECT Clients.nom, Clients.prenom, Achats.produit, Achats.montant
FROM Clients
LEFT JOIN Achats ON Clients.id = Achats.client_id;
```

Utopios® Tous droits réservés

c. RIGHT JOIN :

```
SELECT Clients.nom, Clients.prenom, Achats.produit, Achats.montant
FROM Clients
RIGHT JOIN Achats ON Clients.id = Achats.client_id;
```

d. FULL JOIN :

```
SELECT Clients.nom, Clients.prenom, Achats.produit, Achats.montant
FROM Clients
FULL JOIN Achats ON Clients.id = Achats.client_id;
```

Comme vous pouvez le voir, la syntaxe de chacune des jointures est identique. Mais pas toujours le résultat de son affichage.

Utopios® Tous droits réservés

UTOPIOS®
TRANSFORMATION DIGITALE INTELLIGENTE

Les jointures

Utopios® Tous droits réservés

Introduction

Cette partie est centrée sur les **jointures** qui sont une fonctionnalité avancée du **langage DQL**.

Avant d'aborder le concept de jointure, il est important d'être à l'aise avec le langage DQL, et de savoir créer BDD et tables.

Utopios® Tous droits réservés

Définition

En DQL, nous avons appris à récupérer les données d'**une** table. Mais il est en fait beaucoup plus fréquent que nous ayons à **récupérer les données de plusieurs tables** en même temps au sein d'une seule requête.

Les jointures nous permettent cela, on les trouve principalement sous quatres formes: **INNER JOIN, RIGHT JOIN, LEFT JOIN et FULL JOIN**

Utopios® Tous droits réservés

History_Grades

student_id	student_name	history_grade
1	Alice	A
2	Jane	B
3	Julie	A

Math_Grades

student_id	student_name	math_grade
1	Alice	B
4	Roger	A
5	Kate	C

INNER JOIN

student_id	student_name	history_grade	student_id	student_name	math_grade
1	Alice	A	1	Alice	B

LEFT JOIN

student_id	student_name	history_grade	student_id	student_name	math_grade
1	Alice	A	1	Alice	B
2	Jane	B			
3	Julie	A			

RIGHT JOIN

student_id	student_name	history_grade	student_id	student_name	math_grade
1	Alice	A	1	Alice	B
			4	Roger	A
			5	Kate	C

FULL JOIN

student_id	student_name	history_grade	student_id	student_name	math_grade
1	Alice	A	1	Alice	B
2	Jane	B			
3	Julie	A			
			4	Roger	A
			5	Kate	C

Utopios® Tous droits réservés

Mise en situation

Imaginons que nous sommes dans le contexte d'un site d'abonnements à un service qui propose à ses clients **trois types d'abonnements**:

- L'abonnement STANDARD
- L'abonnement PREMIUM
- L'abonnement VIP

Et nous avons d'autres part **les clients**. Comment rattacher les clients à leurs abonnements ?

Utopios® Tous droits réservés

UTOPIOS®
TRANSFORMATION DIGITALE INTELLIGENTE

Lier les tables: La clé étrangère (Foreign Key)

Utopios® Tous droits réservés

La clé étrangère

Avant de vouloir manipuler nos données, il est important de préparer nos tables à ce qu'elles communiquent entre elles. Ce moyen en SQL, c'est **la clé étrangère** (foreign key).
Elle se précise dès la construction de la table, il est donc préférable d'anticiper l'interaction entre les tables lorsque je construis une base de données.

Utopios® Tous droits réservés

Syntaxe

Pour avoir une jointure, il me faut d'abord deux tables, d'une part, **ma table Clients**:

```
CREATE TABLE Clients (
    id INT PRIMARY KEY AUTO_INCREMENT,
    first_name VARCHAR(100),
    last_name VARCHAR(100),
    city VARCHAR(100),
    age INT
  );
```

Utopios® Tous droits réservés

Syntaxe

D'autre part, **ma table Abonnements**, que je lie à ma table principale (ma table Clients ici):

```
CREATE TABLE Abonnements (
    client_id INT,
    abonnement_type VARCHAR(100),
    FOREIGN KEY (client_id)
    REFERENCES Clients(id)
);
```

Voilà, mes deux tables sont maintenant liées, c'est à dire que je ne pourrai plus insérer dans ma colonne client_id des valeurs qui

Utopios® Tous droits réservés

UTOPIOS®
TRANSFORMATION DIGITALE INTELLIGENTE

INNER JOIN

Utopios® Tous droits réservés

INNER JOIN

Il s'agit de ma première forme de jointure, en français on parle de **jointure intérieure**:

Son rôle est de connecter les lignes de deux tables (ou plus) en se basant sur une colonne qui les relie dans chacune.

Utopios® Tous droits réservés

Syntaxe

```
SELECT Clients.first_name, Clients.last_city, Abonnements.abonnement_type
FROM Clients
INNER JOIN Abonnements
ON Clients.id = Abonnements.client_id;
```

La clause `INNER JOIN` lie les deux tables Clients et Abonnements.
Le mot-clé `ON` est utilisé pour préciser la condition de jonction. Ici, c'est que l'id du client corresponde à celui du client_id de Abonnements.

Utopios® Tous droits réservés

UTOPIOS®
TRANSFORMATION DIGITALE INTELLIGENTE

Les ALIAS de tables

Utopios® Tous droits réservés

Les alias de tables

Lorsqu'on en vient à manipuler plusieurs tables, il devient fastidieux de préciser à chaque fois l'entièreté de la table avant chaque colonne. Mais l'**alias de table** nous permet, comme son nom l'indique, de pouvoir renommer notre table, exactement de la même manière qu'on le faisait pour les colonnes.

```
SELECT c.first_name, c.last_city, a.abonnement_type
FROM Clients AS c
INNER JOIN Abonnements AS a
ON c.id = a.client_id;
```

Utopios® Tous droits réservés

Particularité: Joindre plus de deux tables

```
SELECT c.name, p.product, p.amount
FROM Clients AS c
JOIN Abonnements AS a
ON c.id = a.customer_id
JOIN Products AS p
ON a.product_id = p.product_id;
```

Utopios® Tous droits réservés

UTOPIOS®
TRANSFORMATION DIGITALE INTELLIGENTE

LEFT JOIN/RIGHT JOIN

Utopios® Tous droits réservés

LEFT JOIN/RIGHT JOIN

Une jointure gauche (ou son inverse la jointure droite) va récupérer l'entièreté de la table de gauche et va venir y ajouter les colonnes de la table de droite qui lui sont liées:

On nomme left table: la table qui est mentionné la première dans la requête, la deuxième est la table de droite

Utopios® Tous droits réservés

Syntaxe

```
SELECT *
FROM Clients
LEFT JOIN Abonnements ON Clients.id = Abonnements.client_id;

SELECT *
FROM Clients
RIGHT JOIN Abonnements ON Clients.id = Abonnements.client_id;
```

Si aucune correspondance n'est trouvée dans la table de droite, les colonnes de la table de droite auront des valeurs NULL.

Utopios® Tous droits réservés

UTOPIOS®
TRANSFORMATION DIGITALE INTELLIGENTE

FULL JOIN

Utopios® Tous droits réservés

FULL JOIN

Enfin, la FULL JOIN est simplement la jointure totale de l'ensemble des deux tables

ATTENTION: La clause FULL JOIN n'est pas prise en charge dans toutes les bases de données, notamment MySQL.

Utopios® Tous droits réservés

FULL JOIN en MySQL

En MySQL, on utilisera une combinaison de LEFT JOIN et RIGHT JOIN pour simuler un FULL JOIN grâce à la clause **UNION**.

```
SELECT *
FROM Clients
LEFT JOIN Abonnements ON Clients.id = Abonnements.client_id

UNION

SELECT *
FROM Clients
RIGHT JOIN Abonnements ON Clients.id = Abonnements.client_id
WHERE Clients.id IS NULL;
```

Utopios® Tous droits réservés

Exercice: Découverte des jointures

Création des tables :

- La table "Clients" a les colonnes suivantes : `id` (clé primaire), `nom`, `prenom`, `ville`.
- La table "Achats" a les colonnes suivantes : `client_id` (clé étrangère vers "Clients"), `produit`, `montant`.

Insertion des données :

- Insérez au moins cinq clients dans la table "Clients".
- Insérez au moins dix achats dans la table "Achats", en veillant à ce que certains clients aient effectué des achats et d'autres non.

Utopios® Tous droits réservés

INNER JOIN :

- Sélectionnez les noms et prénoms des clients ainsi que les détails de leurs achats (si disponibles).

LEFT JOIN :

- Sélectionnez tous les clients et les détails de leurs achats s'ils ont effectué des achats, sinon affichez les colonnes des achats avec des valeurs NULL.

RIGHT JOIN :

- Sélectionnez tous les achats et les détails des clients correspondants, même s'il n'y a pas de correspondance.

Utopios® Tous droits réservés

FULL JOIN :

- Sélectionnez tous les clients et tous les achats, en affichant les détails des clients même s'ils n'ont pas effectué d'achats, et vice versa.

Utopios® Tous droits réservés

Correction :

a. INNER JOIN :

```
SELECT Clients.nom, Clients.prenom, Achats.produit, Achats.montant
FROM Clients
INNER JOIN Achats ON Clients.id = Achats.client_id;
```

b. LEFT JOIN :

```
SELECT Clients.nom, Clients.prenom, Achats.produit, Achats.montant
FROM Clients
LEFT JOIN Achats ON Clients.id = Achats.client_id;
```

Utopios® Tous droits réservés

c. RIGHT JOIN :

```
SELECT Clients.nom, Clients.prenom, Achats.produit, Achats.montant
FROM Clients
RIGHT JOIN Achats ON Clients.id = Achats.client_id;
```

d. FULL JOIN :

```
SELECT Clients.nom, Clients.prenom, Achats.produit, Achats.montant
FROM Clients
FULL JOIN Achats ON Clients.id = Achats.client_id;
```

Comme vous pouvez le voir, la syntaxe de chacune des jointures est identique. Mais pas toujours le résultat de son affichage.

Utopios® Tous droits réservés

UTOPIOS®
TRANSFORMATION DIGITALE INTELLIGENTE

Le Data Manipulation Language (DML)

Utopios® Tous droits réservés

Introduction

Ce module est centré sur le **DML** *(Data Manipulation Language)* ou **langage de manipulation de données**
Il s'agit d'un **sous-langage** du langage SQL. Il est, comme son parent, un **langage de requête**.

Ses rôles sont principalement l'**insertion**, la **suppression** et **la modification** des données au sein d'une base. Il permet d'**interagir directement** avec les données dans une BDD établie mais il ne permet ni la création, ni la structuration de nos bases de données en elles-mêmes. (voir sous-langage **DDL**)

Utopios® Tous droits réservés

Avant-Propos

Cette section traite du **langage DML**, il nous permet la modification de données mais **pas la création de tables** dans lesquelles nous pouvons les stocker, rôle qui revient à un autre langage.
Une base de données est donc fournie avec le cours pour permettre la manipulation du langage, elle se prénomme **dql_bdd.sql**
Son implémentation dépendra de l'interface utilisée.

Si vous avez des connaissances sur d'autres sous-langages du SQL, n'hésitez pas à suivre le cours avec vos propres bases de données.

Utopios® Tous droits réservés

Pré-requis

La compréhension et l'utilisation du **langage DQL** est **fortement** recommandé **avant** d'aborder ce cours sur le **DML**. En effet, le DQL nous permet, grâce à la clause **SELECT**, de **voir** les manipulations qui ont été effectuées avec nos clauses de DML.

L'utilisation du DQL n'est pas obligatoire pour utiliser le DML, mais ils sont souvent appris conjointement et le DQL est une étape nécessaire dans notre parcours d'apprentissage pour constater la bonne utilisation de nos clauses.

Utopios® Tous droits réservés

Choisir sa base de données

Avant d'envoyer des requêtes, il me faut choisir dans quelle base de données je souhaite travailler, nos deux premières requêtes seront:

1. Pour afficher la liste des BDD disponibles:

   ```
   SHOW DATABASES;
   ```

2. Pour utiliser celle de mon choix:

   ```
   USE nom_de_la_bdd;
   ```

Utopios® Tous droits réservés

Voir les tables présentes dans la BDD

Une fois que je suis dans la base de données de mon choix, je peux consulter la liste des tables présentes:

- Pour MySQL/MariaDB:

```
SHOW TABLES;
```

Utopios® Tous droits réservés

Le CRUD (Create, Read, Update, Delete)

L'acronyme informatique anglais CRUD désigne les quatre opérations de base pour la persistance des données, en particulier le stockage d'informations en base de données.

Opération	SQL	Action
CREATE	INSERT	Ajouter un/des enregistrement(s)
READ	SELECT	Rechercher
UPDATE	UPDATE	Mettre à jour
DELETE	DELETE	Supprimer

Utopios® Tous droits réservés

UTOPIOS®
TRANSFORMATION DIGITALE INTELLIGENTE

INSERT INTO

Utopios® Tous droits réservés

INSERT INTO...

En SQL, la clause **INSERT INTO** est utilisée pour **ajouter des données** à une table existante.

Elle se construit de la façon suivante:

```
INSERT INTO nom_de_table (colonne1, colonne2, colonne3, ...)
VALUES (valeur1, valeur2, valeur3, ...);
```

Attention: Assurez-vous que les types de données correspondent entre les valeurs insérées et les colonnes de la table.

Utopios® Tous droits réservés

Syntaxes

Insertion de données dans toutes les colonnes :

```
INSERT INTO employees VALUES (1, 'John Doe', 'Manager', 50000);
```

Spécification des colonnes (recommandé) :

```
INSERT INTO employees (employee_id, employee_name, job_title, salary)
VALUES (1, 'John Doe', 'Manager', 50000);
```

Remarque: *Il est possible d'insérer plusieurs lignes d'un coup en les séparant par des virgules*

Utopios® Tous droits réservés

UTOPIOS®
TRANSFORMATION DIGITALE INTELLIGENTE

UPDATE

Utopios® Tous droits réservés

UPDATE

En SQL, la clause **UPDATE** est utilisée pour **mettre à jour des données** dans une table existante.

Elle se construit de la façon suivante:

```
UPDATE nom_de_table
SET colonne1 = valeur1, colonne2 = valeur2, ...
WHERE condition;
```

- **Colonne:** Les colonnes que vous souhaitez mettre à jour.
- **Valeur:** Les nouvelles valeurs que vous souhaitez assigner.
- **WHERE:** La condition pour spécifier les lignes à mettre à jour. Si

Utopios® Tous droits réservés

Exemples

- **Mise à jour de toutes les lignes dans une table :**

```
UPDATE employes
SET salaire = salaire * 1.1;
```

Cela augmente tous les salaires de 10%.

- **Mise à jour avec une condition :**

```
UPDATE employes
SET statut = 'Manager'
WHERE departement = 'Informatique';
```

Utopios® Tous droits réservés

UTOPIOS®
TRANSFORMATION DIGITALE INTELLIGENTE

DELETE

Utopios® Tous droits réservés

DELETE

En SQL, la clause **DELETE** est utilisée pour **supprimer des données** dans une table existante.

Elle se construit de la façon suivante:

```
DELETE FROM nom_de_table
WHERE condition;
```

- nom_de_table: Le nom de la table dont vous souhaitez supprimer des données.
- WHERE: La condition pour spécifier les lignes à supprimer. **Si cette clause est omise, toutes les lignes de la table seront supprimées.**

Utopios® Tous droits réservés

Exemples

- Suppression de toutes les lignes dans une table :

```
DELETE FROM employes;
```

Cela supprime toutes les lignes de la table "employes".

- Suppression avec une condition :

```
DELETE FROM employes
WHERE date_embauche < '2022-01-01';
```

Cela supprime les employés embauchés avant le 1er janvier 2022.

Utopios® Tous droits réservés

DELETE/TRUNCATE

La commande **TRUNCATE** est également allouée à la suppression.

```
TRUNCATE TABLE nom_de_table;
```

Mais, contrairement au DELETE, elle sert généralement à supprimer la **table entière** plutôt qu'une suppression fine via des conditions.

TRUNCATE est plus performant mais également plus dangereux. L'action sera **irréversible**, contrairement au DELETE qu'on peut intégrer au sein de transactions (voir TCL)

Utopios® Tous droits réservés

Exercice: Résumé sur le langage DML

Considérez la table "Students" avec les colonnes suivantes :

```
CREATE TABLE Students (
    student_id INT PRIMARY KEY,
    first_name VARCHAR(50) NOT NULL,
    last_name VARCHAR(50) NOT NULL,
    age INT,
    grade VARCHAR(10)
);
```

Utopios® Tous droits réservés

Exercice: Résumé sur le langage DML

Partie 1: INSERT INTO

Ajoutez trois nouveaux étudiants à la table :

```
Prénom: Maria, Nom: Rodriguez, Âge: 21, Note: B
Prénom: Ahmed, Nom: Khan, Âge: 19, Note: A
Prénom: Emily, Nom: Baker, Âge: 22, Note: C
```

Utopios® Tous droits réservés

Exercice: Résumé sur le langage DML

Partie 2: UPDATE

Mettez à jour la note de l'étudiant ayant le prénom "Maria" pour la changer de "B" à "A".

Augmentez l'âge de tous les étudiants de 1 an.

Utopios® Tous droits réservés

Exercice: Résumé sur le langage DML

Partie 3: DELETE

Supprimez l'étudiant ayant le prénom "Emily" de la table.
Supprimez tous les étudiants dont l'âge est inférieur à 20 ans.

Partie 4: TRUNCATE

Videz complètement la table "Students" en utilisant la commande TRUNCATE.

Utopios® Tous droits réservés

```sql
-- Partie 1: INSERT INTO
INSERT INTO Students (student_id, first_name, last_name, age, grade)
VALUES
    (1, 'Maria', 'Rodriguez', 21, 'B'),
    (2, 'Ahmed', 'Khan', 19, 'A'),
    (3, 'Emily', 'Baker', 22, 'C');

-- Partie 2: UPDATE
UPDATE Students
SET grade = 'A'
WHERE first_name = 'Maria';

UPDATE Students
SET age = age + 1;

-- Partie 3: DELETE
DELETE FROM Students WHERE first_name = 'Emily';
DELETE FROM Students WHERE age < 20;
TRUNCATE TABLE Students;
```

Utopios® Tous droits réservés

UTOPIOS®
TRANSFORMATION DIGITALE INTELLIGENTE

Le Data Definition Language (DDL)

Utopios® Tous droits réservés

Introduction

Ce module est centré sur le **DDL** *(Data Definition Language)* ou **langage de définition de données**
Il s'agit d'un **sous-langage** du langage SQL. Il est, comme son parent, un **langage de requête**.

Il est utilisé pour **définir, modifier et supprimer** la structure des données dans une base de données relationnelle. Son rôle principal est de permettre aux utilisateurs de **spécifier la structure des tables, des vues, des index** et d'autres objets de la base de données.

Utopios® Tous droits réservés

Introduction

Le langage DDL est un langage très étendu qui est à la base de la construction de nombreux schémas de notre BDD. Il va principalement s'articuler autour de **trois commandes centrales:**

CREATE	DROP	ALTER
Création	Suppression	Modification

Utopios® Tous droits réservés

Introduction

Ces précédentes commandes vont avoir une forte influence sur les principales structures de notre BDD. Mais de quelle structure parle-t-on ? Qu'est-il possible de définir au sein de ce langage ?

- Les **databases** elles-mêmes.
- Les **tables** contenues dans ces dbs.
- Les **contraintes d'intégrité** à appliquer à nos tables, pour garantir une cohérence à nos datas.
- Les **index** pour optimiser le tri et la recherche de données.
- Les **vues**, "tables virtuelles" qui affichent sans stocker.

Utopios® Tous droits réservés

UTOPIOS®
TRANSFORMATION DIGITALE INTELLIGENTE

Construire des databases

Utopios® Tous droits réservés

Construire des databases

Nos **trois commandes centrales** vont nous permettre dans un premier temps de construire notre base de données elle-même. Il suffira de lui donner un nom pour pouvoir la créer:

```
-- Créer une base de données
CREATE DATABASE maBaseDeDonnées

-- Supprimer une base de données
DROP DATABASE maBaseDeDonnées
```

Utopios® Tous droits réservés

Convention de nommage des objets

- Ne pas dépasser 128 caractères
- Commencer par une lettre
- Être composé de lettres, de chiffres et du caractère "_"
- Ne pas être un mot réservé du SQL (sauf entre guillemets)
- Ne pas utiliser d'accents
- Utiliser des lettres en minuscules

Ex : client, bdd_ecole, date_inscription

Utopios® Tous droits réservés

Modifier une base de données

La commande **ALTER** fonctionne elle aussi sur la database. Son rôle peut être varié selon le SGBD mais elle sert principalement à **changer le jeu de caractères (character set)** et **modifier la collation** (façon dont les caractères sont triés dans votre BDD)

```
ALTER DATABASE ma_base_de_donnees
  CHARACTER SET utf8mb4
  COLLATE utf8mb4_unicode_ci;
```

Utopios® Tous droits réservés

Vérification d'existence

Une fois qu'on a créé notre BDD, si on souhaite relancer notre script, nous allons être confrontés à une erreur. En effet, la BDD existe déjà. Pour prévenir notre SGBD de ne lancer une requête que dans le cas où elle n'a pas déjà été effectuée, on peut utiliser les clauses **IF EXISTS** et **IF NOT EXISTS**

```
CREATE DATABASE IF NOT EXISTS ma_base_de_donnees
DROP DATABASE IF EXISTS ma_base_de_données
```

Cette commande est valable pour tous les types de structures (tables, vues...)

Utopios® Tous droits réservés

UTOPIOS®
TRANSFORMATION DIGITALE INTELLIGENTE

Construire des tables

Utopios® Tous droits réservés

Construire des tables

Une fois notre base de données définie, et que nous avons spécifié que nous souhaitons travailler à l'intérieur, le moment est venu de **concevoir des tables**.

Les commandes de **création, suppression et modifications** sont relativement proches de celles de la BDD.

Par convention, on va nommer les tables au singulier, sans caractères spéciaux, avec la première lettre en majuscules.

Ex: `Employee`, `Order`, `Product.`

Utopios® Tous droits réservés

Définition de tables

Nous allons donc utiliser la commande `CREATE TABLE` pour créer une table, puis nous allons mettre entre parenthèses **le nom des colonnes** de notre choix suivis de **leur type**. Chaque colonne sera séparée d'une virgule.

```
CREATE TABLE Product (
    name VARCHAR(50),
    quantity INT,
    price DECIMAL,
    send_date DATE
)
```

Utopios® Tous droits réservés

Les types en SQL

En SQL, les types de colonnes **déterminent le type de données pouvant être stocké dans la colonne d'une table**.

Il nous faudra choisir avec précision selon si nous voulons récupérer des nombres, du texte etc…

On peut découper ces types précis en plusieurs grandes familles de types de data.

Nous allons ici parcourir ces différentes familles, mais les types peuvent être très différents d'un SGBD à l'autre.

Utopios® Tous droits réservés

Type de données numériques

Comme leur nom l'indique, ils servent à stocker des nombres. C'est un type de données très commun. En informatique, il est découpé en deux familles **distinctes** :

- **Entiers** : ils stockent des nombres entiers sans partie décimale. Par exemple : INTEGER, INT, SMALLINT, BIGINT.
- **Décimaux** : ils stockent des nombres avec une partie décimale, offrant une précision fixe ou variable. Par exemple : DECIMAL, NUMERIC, FLOAT, DOUBLE.

Utopios® Tous droits réservés

Type de données de chaînes de caractères

- **Chaînes de caractères fixes** : ils stockent des chaînes de longueur fixe. Par exemple : CHAR.
- **Chaînes de caractères variables** : ils stockent des chaînes de longueur variable. Par exemple : VARCHAR, TEXT.

On définira très souvent leur **longueur maximale** entre parenthèses.

Utopios® Tous droits réservés

Types de données temporelles :

- **Date :** ils stockent les valeurs de date (année, mois, jour). Ex: DATE.
- **Heure :** ils stockent des valeurs d'heure. Par exemple : TIME.
- **Date et heure :** Ils stockent à la fois la date et l'heure. Par exemple : DATETIME, TIMESTAMP.

Utopios® Tous droits réservés

Types de données binaires :

- **Blobs :** ils stockent des données binaires de grande taille. Par exemple : BLOB, LONGBLOB.
- **Binaire variable :** ils stockent des données binaires de longueur variable. Par exemple : VARBINARY.

Utopios® Tous droits réservés

Types de données booléennes :

- **Booléens :** ils stockent des valeurs de vérité (VRAI ou FAUX). Par exemple : BOOLEAN, BOOL, BIT.

Types de données géographiques :

- **Géométrie :** ils stockent des données géométriques telles que des points, des lignes, des polygones, etc. Par exemple : GEOMETRY, POINT, LINESTRING, POLYGON.

Utopios® Tous droits réservés

Suppression de tables

De la même manière que pour les bases de données, nous pouvons très facilement **supprimer une table grâce à la commande DROP**, avec ou sans vérification d'existence.

```
-- Suppression classique
DROP TABLE ma_table
-- Avec vérification
DROP TABLE IF EXISTS ma_table
```

Attention, toute suppression de table est **immédiate** et **définitive**. Toutes ses données (si elle en contient) seront perdues.

Utopios® Tous droits réservés

Modification de tables

La modification des tables à l'aide de la clause **ALTER** est très complète :

```
-- Ajout d'une colonne à une table :
ALTER TABLE ma_table
ADD nouvelle_colonne INT;

-- Modification du type de données d'une colonne :
ALTER TABLE ma_table
MODIFY colonne_existante VARCHAR(100);

-- Supression d'une colonne:
ALTER TABLE ma_table
DROP COLUMN colonne_a_supprimer;
```

Utopios® Tous droits réservés

Renommer ses tables et ses colonnes

La clause **ALTER** nous permet également de renommer nos tables et nos colonnes :

```
-- Renommer une table
ALTER TABLE ancien_nom_table
RENAME TO nouveau_nom_table;

-- Renommer une colonne
ALTER TABLE ma_table
CHANGE ancien_nom_colonne nouveau_nom_colonne type_de_donnee;
```

Utopios® Tous droits réservés

UTOPIOS®
TRANSFORMATION DIGITALE INTELLIGENTE

Appliquer des contraintes

Utopios® Tous droits réservés

Appliquer des contraintes

Maintenant que nous avons conçu des tables, il est important de leur appliquer des contraintes. Ces contraintes vont agir comme **des règles ou des restrictions** qu'on appliquerait à nos BDD et qui nous permettent un meilleur filtrage de la data qu'on reçoit.

Les contraintes sont variées, mais leur rôle est important. Elles contribuent à assurer **la cohérence, la validité et la qualité** des données stockées dans la base de données.

Utopios® Tous droits réservés

Liste des contraintes de base

1. **Vérification** (`CHECK`): Permet de spécifier une condition qui doit être respectée pour chaque ligne de la table
2. **Valeur par défaut** (`DEFAULT`): Spécifie une valeur à utiliser lorsqu'aucune valeur n'est fournie lors de l'insertion de données.
3. **Non-nullité** (`NON-NULL`): Spécifie qu'une colonne ne peut pas contenir de valeurs nulles.
4. **Unique** (`UNIQUE`): Assure l'unicité des valeurs dans une colonne ou un ensemble de colonnes.

Utopios® Tous droits réservés

Intégrer des contraintes à nos tables

Vous pouvez intégrer des contraintes à la création d'une table en spécifiant les contraintes directement après la définition de chaque colonne :

```
CREATE TABLE nom_table (
    colonne1 type_de_donnees contrainte,
    colonne2 type_de_donnees contrainte,
);
```

On pourra donc préciser `UNIQUE`, `NOT NULL`, `CHECK` ou `DEFAULT`. **Mais CHECK et DEFAULT nécessitent des valeurs pour être appliquées**

Utopios® Tous droits réservés

Nommer ses contraintes

Une seconde syntaxe existe pour appliquer des contraintes, plus complexe mais elle nous permet de **nommer nos contraintes**.
Il faut pour cela appliquer nos contraintes à la fin de nos colonnes grâce au mot-clé **CONSTRAINT**:

```
CREATE TABLE User (
    id INT
    nom VARCHAR(50) NOT NULL,
    age INT,
    CONSTRAINT check_age CHECK (age >= 18),
);
```

Utopios® Tous droits réservés

Nommer ses contraintes

- **Faciliter la compréhension du schéma** : Les autres développeurs peuvent facilement comprendre l'intention derrière la contrainte.
- **Améliorer la lisibilité du code SQL**
- **Faciliter la maintenance et le dépannage** : Lorsqu'une erreur survient en raison d'une violation de contrainte, un nom significatif peut aider à l'identifier.
- **Simplifier la gestion** : Les noms permettent de la référencer facilement lorsqu'il est nécessaire de la modifier ou de la supprimer ultérieurement.

Utopios® Tous droits réservés

Les propriétés dites "clés"

En SQL, certaines propriétés importantes sont appelés des "clés". Il existe deux clés: **La clé primaire et la clé étrangère**.

- **Clé primaire:** Elle est appelée "clé" car elle est utilisée pour accéder et identifier de manière unique chaque enregistrement dans une table. C'est essentiellement la "clé" qui ouvre la porte à une ligne spécifique dans la table.
- **Clé étrangère:** Elle est appelée "clé" car elle établit une relation entre les tables en reliant les enregistrements d'une table à ceux d'une autre. Elle agit comme une "clé" qui ouvre la porte vers une autre table.

Utopios® Tous droits réservés

La clé primaire

La clé primaire garantit que chaque enregistrement dans une table est **unique** et en garantit son **identification**.

Cela signifie qu'aucun enregistrement **ne peut avoir la même valeur dans la colonne définie comme clé primaire.** Elle aide à garantir l'intégrité des données en **empêchant l'insertion de doublons**. On peut la voir comme une combinaison **NOT NULL + UNIQUE**

```
CREATE TABLE Utilisateur (
    id INT PRIMARY KEY,
    nom VARCHAR(50),
);
```

Utopios® Tous droits réservés

l'Auto-incrémentation/identity

Garder unique une clé primaire peut parfois être fastidieux lorsque c'est réalisé manuellement. C'est pour cela que la plupart des SGBD proposent des fonctions d'auto-incrémentation (**AUTO_INCREMENT** en MySQL, **SERIAL** en PostgreSQL, **IDENTITY** en SQL Server.)

Lorsque vous définissez une colonne avec la propriété d'auto-incrémentation, le SGBD se charge **automatiquement** d'attribuer une valeur à cette colonne lors de l'insertion de nouvelles lignes dans la table, **garantissant ainsi l'unicité des valeurs générées.**

Utopios® Tous droits réservés

La clé étrangère

Elle a un rôle bien différent, elle sert de **connexion vers une autre table de notre BDD**.

Par exemple, dans une table de commandes, la clé étrangère peut être l'identifiant de l'employé, qui fait référence à la clé primaire de la table des employés. Cela établit un lien entre chaque commande et l'employé qui l'a passée.

Utopios® Tous droits réservés

La clé étrangère - Syntaxe

```
CREATE TABLE Commande (
    id INT PRIMARY KEY,
    id_client INT,
    date_commande DATE,
    FOREIGN KEY (id_client) REFERENCES Client(id)
);
```

Dans cet exemple :

- Nous créons une table Commande.
- La colonne id_client dans la table Commande fait référence à la colonne id dans la table Client.

Utopios® Tous droits réservés

Ajout de clé à une table

Bien sûr, comme les autres contraintes, les clés peuvent également être ajoutées après la création d'une table grâce à la clause **ALTER**.

```
-- Exemple avec une table "Utilisateurs"
ALTER TABLE Utilisateurs
ADD CONSTRAINT pk_utilisateur PRIMARY KEY (id);

-- Ou avec une table "Commandes"
ALTER TABLE Commandes
ADD CONSTRAINT fk_commandes FOREIGN KEY (id_client) REFERENCES Clients(id);
```

Utopios® Tous droits réservés

UTOPIOS®
TRANSFORMATION DIGITALE INTELLIGENTE

Les views

Utopios® Tous droits réservés

Les views

Les vues sont des **tables virtuelles** dérivées à partir d'autres tables (appelées tables de base). Elles ne sont jamais stockées sur disque, mais maintenues en **mémoire vive**.

Les vues sont des **requêtes SQL pré-compilées**. Elles permettent d'éviter l'exécution de **requêtes coûteuses** à répétition.

Elles permettent de simplifier la complexité des requêtes en **encapsulant une logique de requête complexe dans un objet virtuel facile à utiliser.**

Utopios® Tous droits réservés

Lorsqu'une vue est créée, son contenu est constamment **maintenu à jour** lors des opérations du DML sur les tables de base concernées.

Une vue peut servir de base à une autre vue.

Utopios® Tous droits réservés

Avantages des vues

- Simplification des Requêtes : Les vues permettent d'encapsuler des requêtes complexes, ce qui simplifie le code utilisé dans les applications.

- Sécurité des Données : Les vues peuvent être utilisées pour restreindre l'accès aux données en limitant les accès utilisateurs.

Utopios® Tous droits réservés

Avantages des vues

- **Abstraction des Données :** Les vues permettent d'abstraire la structure sous-jacente des tables, ce qui facilite la modification de la structure sans avoir à modifier les applications qui utilisent la vue.

- **Réutilisation du Code :** Les vues peuvent être utilisées pour réutiliser du code SQL commun dans plusieurs requêtes, ce qui réduit la duplication du code.

Utopios® Tous droits réservés

Utilisation courante des vues

- **Sélection de Données :** Les vues peuvent être utilisées pour sélectionner des données à partir d'une ou plusieurs tables de manière simple et efficace.

- **Modification de Données :** Dans certains systèmes de gestion de base de données, il est possible de modifier les données à travers une vue, bien que cela dépende des fonctionnalités spécifiques du SGBD.

- **Jointures :** Les vues peuvent être utilisées pour simplifier les opérations de jointure en encapsulant la logique de jointure dans une vue.

Utopios® Tous droits réservés

Création d'une vue

Pour créer une vue (view), il suffit, comme pour la plupart des opérations en DDL, d'utiliser le mot-clé `CREATE`, ensuite on indique ce à quoi elle correspond grâce au mot-clé `AS`.

```
CREATE VIEW nom_vue AS
SELECT colonne1, colonne2, ...
FROM table
WHERE condition;
```

Utopios® Tous droits réservés

Si les colonnes résultantes sont issues de calculs ou de fonctions d'agrégation, il faut nommer les colonnes de la vue.

```
CREATE VIEW EmployeDepartement AS
SELECT
  emp.Nom AS Nom, emp.Prenom AS Prenom, dep.Nom AS Departement
FROM
  Employe AS emp INNER JOIN Departement AS dep ON emp.IdDepartement = dep.Id
WHERE
  DATEDIFF(NOW(), emp.DateNaissance)/ 365 > 35;
-- une vue ayant le nom, le prénom et le département des employés de plus de 35 ans
```

Utopios® Tous droits réservés

```sql
CREATE VIEW DepartementInfo(Nom, NbrEmploye, SalaireTotal) AS
SELECT
  dep.Nom,
  COUNT(*),
  SUM(emp.Salaire)
FROM
  Employe AS emp,
  Departement AS dep
WHERE
  emp.IdDepartement = dep.Id
ORDER BY
  dep.Id;
-- une vue sur les statistiques des départements (nom, nombre d'employés et salaire total)

DROP VIEW EmployeDepartement, DepartementInfo;
```

Utopios® Tous droits réservés

Suppression de vues

La commande **DROP VIEW** nous permet de supprimer une vue lorsqu'elle ne nous est plus utile.

```
DROP VIEW nom_vue;
```

C'est une manipulation importante car

- Les vues consomment des ressources lorsqu'elles sont créées et stockées en mémoire.
- Les vues peuvent être utilisées pour accéder aux données sensibles d'une BDD. Si des vues inutilisées existent et ne sont pas correctement sécurisées, cela peut représenter un risque en plus.

Utopios® Tous droits réservés

Toutes les opérations du DML s'appliquent sur les vues (consultation, insertion, modification ou suppression).

Néanmoins, certaines opérations du DML sur une vue peuvent être très complexes et ambiguës. Par exemple, si la vue est créée à partir de plusieurs tables, plusieurs interprétations peuvent être faites.

Selon leur définition, certaines vues ne peuvent être mises à jour car de telles requêtes ne font aucun sens (voir le dernier exemple).

Utopios® Tous droits réservés

UTOPIOS®
TRANSFORMATION DIGITALE INTELLIGENTE

L'indexation

Utopios® Tous droits réservés

Les indexes permettent une **amélioration notable des performances** des SGBD.

Ils ne sont pas prescrits par la norme SQL, mais font partie intégrante de toutes les implémentations des SGBDR.

Leur usage est très simple : le DDL permet de définir ou de supprimer les indexes. Toutes les opérations du DML gèrent automatiquement l'utilisation adéquate des indexes sans intervention du concepteur.

Néanmoins, une optimisation de la base de données nécessite une analyse permettant de faire une **définition adéquate** des indexes.

Utopios® Tous droits réservés

Chaque SGBD étant différent, nous verrons ici l'utilisation avec MySQL. Oracle présente des concepts similaires mais offre une plus grande panoplie d'outils.

Un index peut être créé sur plusieurs colonnes à la fois.

Chaque clé primaire crée automatiquement un index de type unique.

```
CREATE [UNIQUE | FULLTEXT | SPATIAL] INDEX nom_index
ON nom_table( nom_colonne1 [(longueur_champ)] [ASC | DESC], ...) [ USING {BTREE | HASH} ];

DROP INDEX nom_index ON nom_table;
```

Utopios® Tous droits réservés

UTOPIOS®
TRANSFORMATION DIGITALE INTELLIGENTE

Les requêtes préparées

Utopios® Tous droits réservés

Les requêtes préparées (également appelées requêtes paramétrisées, ou « prepared statements » en anglais) sont des requêtes stockées et précompilées par le SGBD. Ces requêtes **ne sont pas exécutées** lors de leur déclaration.

Via un mécanisme de liaison (« binding »), elles offrent un outil puissant permettant de définir les valeurs des paramètres de la requête lors de la demande de leur exécution.

En effet, il est fréquent de faire la même requête en ne faisant varier que certains paramètres.

Utopios® Tous droits réservés

```
SELECT Nom, Prenom FROM Employe WHERE Departement = 'Ventes';
SELECT Nom, Prenom FROM Employe WHERE Departement = 'Achats';
```

Pour un SGBD, les deux requêtes précédentes sont différentes, et demandent une interprétation indépendante par le SGBD lors de leur exécution.

En prenant cet exemple, si cette requête est fréquente, il sera préférable d'utiliser le mécanisme des requêtes préparées. Ainsi, une seule analyse est faite lors de la déclaration, et tous les appels subséquents pourront avoir leur propre spécificité.

Utopios® Tous droits réservés

L’usage d'une requête préparée se fait en trois étapes :

1. Préparation
   - on définit la requête préparée dans le SGBD
   - le SGBD analyse la requête et initialise les ressources internes nécessaires à son exécution

Utopios® Tous droits réservés

2. Liaison et exécution
   - le client envoie les paramètres de la requête
   - le SGBD lie les paramètres et exécute la requête avec les ressources préalablement allouées
3. Libération des ressources
   - lorsque la requête préparée n'est plus nécessaire, il est important de libérer les ressources.

Utopios® Tous droits réservés

Les requêtes préparées ne font pas partie de la norme SQL, mais sont des outils supplémentaires offerts par les SGBD.

Voici les éléments syntaxiques pour MySQL :

```
PREPARE nom_requete_prep FROM requete_txt;
EXECUTE nom_requete_prep [USING @variable1, @variable2, ...];
[DEALLOCATE | DROP] PREPARE nom_requete_prep ;
```

Utopios® Tous droits réservés

Le paramètre requete_txt doit être une requête mise sous forme de chaîne de caractères. Tous les paramètres devant être liés plus tard doivent être identifiés par le caractère « ? ».

Les variables passées à l'appel de `EXECUTE` doivent être présentées dans le même ordre que leur emplacement dans la requête (caractère « `?` »).

Utopios® Tous droits réservés

Il est possible de déclarer et d'assigner des variables sessions avec la clause `SET` de la façon suivante :

```
PREPARE EmpParDepartement FROM 'SELECT Nom, Prenom
    FROM Employe
    WHERE Departement = ? AND Salaire > ?';

SET @dep = 'Ventes', @sal = 75000;
EXECUTE EmpParDepartement USING @dep, @sal;

SET @dep = 'Achats', @sal = 85000;
EXECUTE EmpParDepartement USING @dep, @sal;

DEALLOCATE PREPARE EmpParDepartement;
```

Utopios® Tous droits réservés

Attention, il existe plusieurs restrictions aux requêtes préparées :

- les mots réservés du langage SQL : `SELECT`, `FROM`, `CREATE`, etc.
- les opérateurs : `=`, `<`, `>=`, `AND`, etc.
- Les fonctions : `NOW`, `UPPER`, etc.
- Les objets de la base de données (aucune table, vue, index ou autre).

Utopios® Tous droits réservés

UTOPIOS®
TRANSFORMATION DIGITALE INTELLIGENTE

Les procédures stockées

Utopios® Tous droits réservés

Les procédures stockées

Une procédure stockée est un ensemble d'**instructions précompilées**.

Comme pour les requêtes préparées, les procédures stockées sont interprétées, analysées, planifiées et optimisées lors de leur déclaration et rendues disponibles pour une **exécution rapide** et flexible via des paramètres définis en entrée et en sortie.

Utopios® Tous droits réservés

Les requêtes préparées et les procédures stockées ont beaucoup d'avantages en commun. Néanmoins, les mécanismes sous-jacents sont très différents.

Les procédures stockées ne font pas partie de la norme SQL, ils sont plutôt des outils supplémentaires offerts par les SGBD.

Les SGBD qui permettent les procédures stockées mettent à disposition un langage procédural complémentaire permettant de plus vastes possibilités que le simple SQL. Oracle offre le langage PL/SQL, tandis que MySQL offre un langage propriétaire très similaire. Nous verrons celui de MySQL.

Utopios® Tous droits réservés

Les procédures stockées

Avantages :

- simplification pour les développeurs
- rapidité par la réduction de la bande passante
- gain de performance puisque déjà préparées lors de leur déclaration
- gain de performance puisqu'exécutées du côté serveur
- accroissement de la sécurité en limitant l'accès aux objets définis uniquement dans les procédures stockées

Utopios® Tous droits réservés

Inconvénients :

- développement et maintenance plus difficiles
- peu d'outil de déverminage pour aider au développement (MySQL n'offre rien)
- peut engendrer une surcharge de calcul du côté serveur

La présentation qui suit prend pour acquis que le lecteur a déjà des connaissances de base avec un langage de programmation procédurale tel que le langage C

Utopios® Tous droits réservés

Principaux éléments syntaxiques

Déclaration, appel et suppression d'une procédure stockée :

```
DELIMITER // -- Ne pas oublier l'instruction DELIMITER

CREATE PROCEDURE nom_procedure[(paramètres)]
BEGIN
contenu_procedure
END

DELIMITER ;
CALL nom_procedure;

DROP PROCEDURE nom_procedure;
```

Utopios® Tous droits réservés

Variables locales : déclaration et affectation

Les variables sont créées et libérées avec la fonction :

```
DECLARE nom_variable TYPE(option) [DEFAULT valeur];
SET nom_variable = valeur;
SELECT colonne INTO nom_variable FROM nom_table ...;
```

Les variables de session sont disponibles pendant toute la session de l'usager et ne sont pas liées à une procédure en particulier. Elles sont précédées par le caractère `@`.

Utopios® Tous droits réservés

Il existe trois types de paramètres pouvant être utilisés avec les procédures stockées : `IN`, `OUT` et `INOUT`

- `IN` : ce type de paramètre sert à passer une valeur à la procédure. La modification de ce paramètre à l'intérieur de la procédure n'a aucun effet sur la valeur externe. Il s'agit du mode **par défaut**.
- `OUT` : ce type de paramètre permet à la procédure de retourner une valeur au programme appelant. La valeur initiale de ce paramètre est inaccessible.
- `INOUT` : combinaison des deux types précédents.

Utopios® Tous droits réservés

Exemple :

```
CREATE PROCEDURE NomDepartement(IN noDep NUMERIC, OUT nomDep VARCHAR(32))
    BEGIN
        SELECT nom INTO nomDep FROM Departement WHERE id = noDep;
    END
```

Utopios® Tous droits réservés

Structures conditionnelles IF et CASE

```
IF condition THEN instructions
    [ELSEIF condition THEN instructions]
    [ELSE instructions]
END IF

CASE expression
    WHEN expression_case_1 THEN instructions_1
    WHEN expression_case_2 THEN instructions_2
    ...
    ELSE instructions
END CASE
```

Utopios® Tous droits réservés

Boucles WHILE, REPEAT et LOOP

```
WHILE condition DO
    instructions
END WHILE
REPEAT
    instructions
UNTIL expression
END REPEAT

identifiant_loop: LOOP
    instructions
    [LEAVE identifiant_loop;]        -- équivalent au break en langage C
    [ITERATE identifiant_loop;]      -- équivalent au continue en langage C
END LOOP
```

Utopios® Tous droits réservés

Le curseur et les procédures stockées

Un curseur est un outil important des SGBDR. Ils permettent le parcours ligne par ligne du contenu d'une table. On peut ainsi faire un traitement spécifique pour chacune des lignes.

Les curseurs ont une syntaxe plus exigeante et une lecture plus approfondie de ce sujet est nécessaire pour une compréhension complète. Néanmoins, voici les éléments importants.

Les curseurs s'utilisent à l'intérieur d'une procédure stockée

Utopios® Tous droits réservés

Pour MySQL, les curseurs ont ces propriétés :

- ils sont en lecture seule (on ne peut donc modifier directement une entrée dans la table)
- le parcours se fait uniquement dans l'ordre de la requête utilisée
- ils travaillent directement sur les données sources (plus rapide mais sensible aux tâches parallèles qui peuvent modifier les données)

Utopios® Tous droits réservés

L'usage d'un curseur se fait en quatre grandes étapes :

1. Déclaration du curseur et de la condition d'arrêt
2. Ouvrir le curseur
3. Accéder aux lignes une à la fois dans une boucle de contrôle
4. Fermer le curseur

Utopios® Tous droits réservés

```
-- Étape 1a : déclaration du curseur
DECLARE nom_curseur CURSOR FOR requete_standard;
-- Étape 1b : déclaration de la var. de contrôle et de la condition d'arrêt
DECLARE variable_controle INTEGER DEFAULT valeur_depart;
DECLARE CONTINUE HANDLER FOR NOT FOUND SET variable_controle = valeur_arrêt;

-- Étape 2 : ouverture du curseur
OPEN nom_curseur;

-- Étape 3 : parcourir les données
label_loop: LOOP;
FETCH nom_curseur INTO variable_reception1 [,variable_reception2 ...]
IF variable_controle = valeur_arrêt THEN LEAVE label_loop;
END IF;
-- faire quelque chose
END LOOP label_loop;
-- Étape 4 : fermer curseur
CLOSE label_loop;
```

Utopios® Tous droits réservés

Procédure retournant une liste de courriels pour l'envoi à toutes les femmes de l'entreprise

```
DELIMITER //
CREATE PROCEDURE ListeCourriel(OUT liste TEXT)
    BEGIN
        -- variables, puis cursor, puis handler
        DECLARE finParcours INT DEFAULT 0;
        DECLARE empGenre CHAR(1);
        DECLARE empCourriel VARCHAR(64);
        DECLARE listeTemp AS TEXT DEFAULT "";
        DECLARE curseurEmploye CURSOR FOR
                        SELECT Genre, Courriel FROM Employe;
        DECLARE CONTINUE HANDLER FOR NOT FOUND SET finParcours = 1;

        OPEN curseurEmploye;
```

Utopios® Tous droits réservés

```
        parcoursEmploye: LOOP
            FETCH curseurEmploye INTO empGenre, empCourriel
            IF finParcours = 1 THEN
                LEAVE parcoursEmploye;
            END IF;

            IF empGenre = 'F' THEN
                SET listeTemp = CONCAT(listeTemp, empCourriel, ";");
            END IF;
        END LOOP parcoursEmploye;

        SET liste = listeTemp;
        CLOSE curseurEmploye ;
    END //
DELIMITER ;
```

Utopios® Tous droits réservés

Plusieurs autres éléments pourraient être couverts concernant les procédures stockées.

Par exemple, la gestion d'erreur et l'émission de condition d'erreur sont des outils puissants permettant une gestion efficace et serrée du SGBD.

Utopios® Tous droits réservés

En lien avec les procédures stockées, il est possible d'écrire des **fonctions stockées** qui sont pratiquement identiques aux procédures.

La principale différence est qu'une fonction retourne une et une seule valeur. Par conséquent, tous les paramètres de la fonction sont implicitement identifiés comme étant des paramètres de type `IN`.

De plus, les fonctions peuvent être appelées à n'importe quel endroit sans nécessité de l'instruction `CALL`.

Utopios® Tous droits réservés

```
CREATE FUNCTION nom_function(param1 [, param2 ...]) RETURNS type_de_retour
BEGIN
-- ...
RETURN valeur_retournée;
END
```

```
DELIMITER // CREATE FUNCTION initialCompact(
  Nom VARCHAR(32),
  Prenom VARCHAR(32)
) RETURNS CHAR(2) BEGIN RETURN UPPER(
  CONCAT(LEFT(Prenom, 1),LEFT(Nom, 1))
);
END // DELIMITER;
SELECT
  initialCompact('Martin', 'Bob');      -- retourne : BM
```

Utopios® Tous droits réservés

UTOPIOS®
TRANSFORMATION DIGITALE INTELLIGENTE

Événements et déclencheurs

Utopios® Tous droits réservés

Les déclencheurs

Les déclencheurs (`TRIGGERS` en anglais) sont des objets de la base de données permettant d'exécuter des tâches spécifiques à des moments spécifiques.

Les événements pouvant être associés à un `TRIGGER` sont principalement liés aux opérations de manipulation des données : `INSERT`, `UPDATE`, `DELETE`.

Utopios® Tous droits réservés

Avantages :

- permet une solution avancée d'assurer l'intégrité référentielle;
- permet d'identifier des erreurs logiques lors des transactions liées au DML
- les événements permettent d'exécuter des tâches planifiées (maintenance, gestion, etc.)
- outils très puissants pour gérer les historiques de transaction liées au DML

Utopios® Tous droits réservés

Inconvénients :

- limitation de certains SGBD sur les réelles possibilités apportées par le concept
- les déclencheurs sont invisibles aux applications clientes et ainsi il peut être difficile de tracer la ligne entre ce qui doit être traité côté serveur / côté client
- les déclencheurs exigent des ressources significatives
- les déclencheurs complexifient le schéma et les interrelations de la BDD. Son développement et sa maintenance peuvent être très coûteuses.

Utopios® Tous droits réservés

Pour le langage SQL, voici les éléments syntaxiques importants :

```
CREATE TRIGGER nom_trigger
    { BEFORE | AFTER } { INSERT | UPDATE | DELETE }
    ON nom_table FOR EACH ROW
    BEGIN
    -- liste d'instructions
    END
```

`BEFORE` et `AFTER` indique que l'évènement est appelé avant ou après l'instruction qui peut être ces éléments du DML `INSERT`, `UPDATE` et `DELETE`.

Utopios® Tous droits réservés

À l'intérieur du `TRIGGER`, les mots-clés `OLD` et `NEW` vous permettent d'accéder aux colonnes des lignes affectées par un déclencheur. `OLD` fait référence à la ligne antérieurement inscrite dans la table. `NEW` fait référence aux nouvelles données.

```
CREATE TABLE Vehicule (
    Id  INTEGER PRIMARY KEY, Marque     VARCHAR(32),
    Modele      VARCHAR(32), Annee      DECIMAL(4,0),
    Etat        ENUM('Disponible', 'Réparation', 'Loué', 'Indisponible')
);
CREATE TABLE HistoriqueLocationVehicule (
    Id  SERIAL PRIMARY KEY,
    IdVehicule INTEGER,
    Etat        ENUM('Disponible', 'Réparation', 'Loué', 'Indisponible'),
    DateAction TIMESTAMP,
    FOREIGN KEY (IdVehicule) REFERENCES Vehicule(Id)
);
```

Utopios® Tous droits réservés

```
DELIMITER //
CREATE TRIGGER mise_a_jour_historique
AFTER UPDATE ON Vehicule FOR EACH ROW
    BEGIN
        IF OLD.Etat <> NEW.Etat THEN
            INSERT INTO HistoriqueLocationVehicule(IdVehicule, Etat, DateAction)
            VALUES (NEW.Id, NEW.Etat, CURTIME());
    END IF;
DELIMITER ;
END //


INSERT INTO Vehicule(Id, Marque, Modele, Annee)
VALUES(77, 'Ford', 'Mustang', 2013);
UPDATE Vehicule SET Etat = 'Réparation' WHERE Vehicule.Id = 77;
UPDATE Vehicule SET Etat = 'Disponible' WHERE Vehicule.Id = 77;
UPDATE Vehicule SET Etat = 'Loué' WHERE Vehicule.Id = 77;
UPDATE Vehicule SET Etat = 'Disponible' WHERE Vehicule.Id = 77;
```

Utopios® Tous droits réservés

Pour MySQL, il est aussi possible de créer des `EVENT` basés sur le temps.

La différence majeure entre les `TRIGGER` et les `EVENT` est que les `TRIGGER` sont synchrones (`BEFORE INSERT`, `AFTER INSERT`, etc.) alors que les `EVENT` sont asynchrones.

Un exemple simple d'événement :

```
CREATE EVENT exemple_event
ON SCHEDULE AT '2006-02-10 23:59:00' DO
INSERT INTO table_du_temps VALUES (NOW());
```

Utopios® Tous droits réservés

UTOPIOS®
TRANSFORMATION DIGITALE INTELLIGENTE

Les transactions

Utopios® Tous droits réservés

Les transactions

Les propriétés d'une transaction sont les suivantes :

- atomicité : toutes les opérations unitaires nécessaires à une transaction sont réalisées (sinon, on annule les opérations intermédiaires)
- consistance : la base de données reste consistante après chaque transaction validée
- isolation : chaque transaction est invisible pour les autres, et assure un fonctionnement indépendant de ces dernières
- durabilité : assure la persistance des transactions validées.

Utopios® Tous droits réservés

Les transactions sont un concept important de tous les SGBD liés au langage SQL.

Pour MySQL, on retrouve principalement 5 instructions liées au TCL (Transaction Control Language) :

- `SAVEPOINT` : crée un point de sauvegarde intermédiaire dans la transaction;
- `COMMIT` : valide la transaction en cours et sauvegarde les modifications faites;
- `ROLLBACK` ou `ROLLBACK TO <savepoint>`: annule la transaction en cours et annule les modifications faites;

Utopios® Tous droits réservés

- `SET autocommit` = { 0 | 1 };
  active ou désactive la validation automatique (par défaut, autocommit = 1);
- `START TRANSACTION` : permet la déclaration du début d'une transaction et certaines options .

Dès qu'une instruction `COMMIT` ou `ROLLBACK` a été exécutée, une nouvelle transaction débute et terminera au prochain appel du `COMMIT` ou `ROLLBACK`.

Utopios® Tous droits réservés

Quelques précisions sur le déroulement du `START TRANSACTION` :

- si il y a une transaction en cours, un `COMMIT` est automatiquement appliqué pour terminer la transaction antérieure;
- si la validation automatique est activée, elle est temporairement désactivée pendant la transaction;
- `START TRANSACTION` définit le point de départ de la transaction et applique les options;
- la transaction a cours jusqu'au prochain `COMMIT` ou `ROLLBACK`;
- puis, si la validation automatique avait été suspendue, elle est réactivée;

Utopios® Tous droits réservés

```
INSERT ... -- operation 1 : sauvegardée autocommit par défaut
SET autocommit = 0;
INSERT ... -- operation 2 : début d'une transaction
INSERT ... -- operation 3 : poursuite de la transaction
COMMIT; -- fin de la transaction, les opérations 2 et 3 sont sauvegardées
UPDATE ... -- operation 4 : début d'une nouvelle transaction
DELETE ... -- operation 5 : poursuite de la transaction
ROLLBACK; -- fin de la transaction et les opérations 4 et 5 sont annulées
```

Utopios® Tous droits réservés

```
START TRANSACTION; -- début d'une transaction, toutes les operations
-- antérieures sont sauvegardées (COMMIT)
INSERT ... -- operation 1 : poursuite de la transaction
DELETE ... -- operation 2 : poursuite de la transaction
SAVEPOINT okTemp; -- définie un point de reprise
INSERT ... -- operation 3 : poursuite de la transaction
DELETE ... -- operation 4 : poursuite de la transaction
ROLLBACK TO okTemp; -- Annule les operations 3 et 4.
INSERT ... -- operation 5 : poursuite de la transaction
DELETE ... -- operation 6 : poursuite de la transaction
IF ... THEN COMMIT -- fin de la trans. et sauvegarde op. 1, 2, 5 & 6
ELSE ROLLBACK END IF; -- fin de la trans. et annule les op. 1, 2, 5 & 6
```

Utopios® Tous droits réservés

UTOPIOS®
TRANSFORMATION DIGITALE INTELLIGENTE

Utilisateurs, rôles et permissions

Utopios® Tous droits réservés

Gestion des utilisateurs

MySQL permet la création de comptes qui autorisent les utilisateurs clients à se connecter au serveur et à accéder aux données gérées par le serveur.

La fonction principale du système de privilèges MySQL est d'**authentifier** un **utilisateur** qui se connecte à partir d'un **hôte** donné et d'associer à cet utilisateur des **privilèges** sur une base de données tels que `SELECT`, `INSERT`, `UPDATE` et `DELETE`.

Utopios® Tous droits réservés

La gestion des utilisateurs consiste en des instructions SQL telles que `CREATE USER`, `GRANT` et `REVOKE`.

Les informations des utilisateurs MySQL sont stockées dans les tables du schéma système mysql.

Utopios® Tous droits réservés

Créer un utilisateur

Pour chaque compte, `CREATE USER` crée une nouvelle ligne dans la table système mysql.user

Lorsqu'il est créé pour la première fois, un compte n'a aucun privilège et son rôle par défaut est NONE

```
CREATE USER '<nom>'@'localhost'
IDENTIFIED BY '<mot de passe>';
```

Utopios® Tous droits réservés

Accorder des droits aux utilisateurs

L'instruction `GRANT` permet d'accorder des privilèges ou des rôles à des comptes.

L'instruction `REVOKE` permet de révoquer les privilèges accordés.

`SHOW GRANTS` permet d'afficher les privilèges dont dispose un compte.

```
GRANT
    priv_type [(column_list)]
      [, priv_type [(column_list)]] ...
    ON [object_type] priv_level
    TO user_specification [ user_options ...]
```

Utopios® Tous droits réservés

Exemple d'utilisation de GRANT

```
-- Donner accès en lecture à la colonne nom et prénom de la table employe
GRANT SELECT (nom, prenom) on employe to 'jean'@'localhost';

-- Donner tous les droits sur la base de données (root)
GRANT ALL PRIVILEGES ON * . * TO 'toto'@'localhost' IDENTIFIED BY 'password';

SHOW GRANTS FOR 'toto'@'localhost';
```

Utopios® Tous droits réservés

Créer des rôles

L'instruction `CREATE ROLE` crée un ou plusieurs rôles

La longueur maximale d'un rôle est de 128 caractères

```
CREATE [OR REPLACE] ROLE [IF NOT EXISTS] role
  [WITH ADMIN
    {CURRENT_USER | CURRENT_ROLE | user | role}]
```

Utopios® Tous droits réservés