SGBDR & STRUCTURE DE LA DONNÉES

INTRODUCTION AU SGBDR ET MODÉLISATION

INTRODUCTION

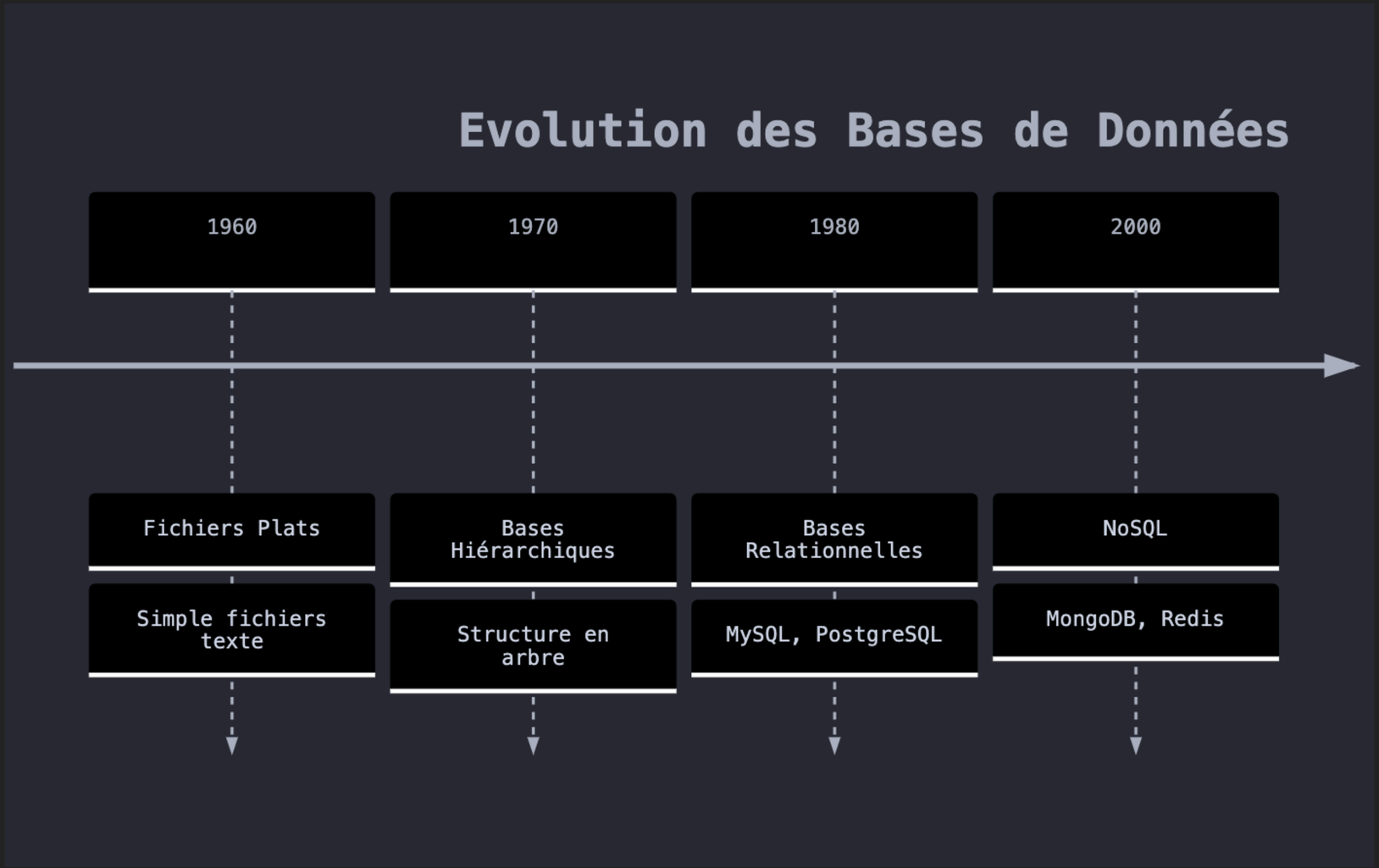
DÉFINITION D'UNE BASE DE DONNÉES

- Une base de données est un ensemble structuré de données organisées de manière à être facilement consultables, gérables et mises à jour
- Elle permet de stocker et de retrouver l'intégralité des données liées à une activité spécifique

POURQUOI UTILISER UN SGBDR ?

- Gestion centralisée des données
- Intégrité et cohérence des données
- Accès concurrent aux données
- Sécurité et contrôle des accès
- Sauvegarde et restauration facilitées
- Réduction de la redondance des données
- Performance optimisée pour les requêtes complexes

HISTORIQUE RAPIDE



SGBDR POPULAIRES

MySQL

- Open source
- Très populaire pour les applications web
- Facile à prendre en main

SGBDR POPULAIRES

MariaDB

- Fork open source de MySQL créé par les développeurs originaux de MySQL
- Meilleure performance (notamment pour les requêtes complexes)
- Facile à prendre en main

SGBDR POPULAIRES

PostgreSQL

- Open source
- Fonctionnalités avancées
- Excellent support des standards SQL

SGBDR POPULAIRES

Oracle

- Leader du marché entreprise
- Très robuste et scalable
- Nombreuses fonctionnalités avancées

SGBDR POPULAIRES

SQL Server

- Solution Microsoft
- Bonne intégration avec l'écosystème Windows
- Outils de BI intégrés

SGBDR POPULAIRES

SQL Server

- Solution Microsoft
- Bonne intégration avec l'écosystème Windows
- Outils de BI intégrés

CONCEPTS FONDAMENTAUX

DONNÉES STRUCTURÉES VS DONNÉES NON STRUCTURÉES

Données structurées :

- Format prédéfini
- Organisation en tables
- Facilement interrogeables
- Exemple : tableaux Excel, bases de données relationnelles

DONNÉES STRUCTURÉES VS DONNÉES NON STRUCTURÉES

Données non structurées :

- Pas de format prédéfini
- Texte libre, images, vidéos
- Plus difficiles à analyser
- Exemple : emails, documents Word

LE MODÈLE RELATIONNEL

- Basé sur la théorie des ensembles
- Données organisées en tables (relations)
- Relations entre les tables via des clés
- Manipulation des données via des opérations relationnelles

TABLES, COLONNES, LIGNES

- Table : représentation d'une entité
- Colonne : attribut de l'entité
- Ligne : occurrence ou instance de l'entité
- Exemple :

```
Table UTILISATEUR
ID | NOM     | EMAIL             | AGE
1  | Jean    | jean@email.com    | 25
2  | Marie   | marie@email.com   | 30
```

TYPES DE DONNÉES COURANTES

Numériques

- INTEGER : nombres entiers
- DECIMAL/NUMERIC : nombres décimaux
- FLOAT/DOUBLE : nombres à virgule flottante

TYPES DE DONNÉES COURANTES

Textuels

- CHAR : chaînes de caractères de taille fixe
- VARCHAR : chaînes de caractères de taille variable
- TEXT : texte long

TYPES DE DONNÉES COURANTES

Dates et Temps

- DATE : date
- TIME : heure
- DATETIME/TIMESTAMP : date et heure

TYPES DE DONNÉES COURANTES

Autres

- BOOLEAN : vrai/faux
- BLOB : données binaires
- ENUM : liste de valeurs prédéfinies

INSTALLATION ET CONFIGURATION

SUR WINDOWS

Téléchargement

- Rendez-vous sur mariadb.org/download
- Téléchargez la dernière version stable (actuellement 11.2.x)
- Choisissez le MSI Installer pour Windows
- Exécutez le fichier .msi et suivez l’assistant d’installation

LINUX

```
# Mettre à jour les paquets
sudo apt update
sudo apt upgrade

# Installer MariaDB
sudo apt install mariadb-server mariadb-client

# Démarrer le service
sudo systemctl start mariadb

# Activer le démarrage automatique
sudo systemctl enable mariadb
```

MAC

- Installez Homebrew si nécessaire
- Puis tapez la commande brew install mariadb dans un terminal
- Pour démarrer le service tapez la commande brew services start mariadb

CONFIGURATION INITIALE

Lancement du script de configuration :

```
sudo mariadb-secure-installation
```

Le terme sudo est à utiliser seulement pour les utilisateurs de Linux et Mac.

CONFIGURATION INITIALE

```
Enter current password for root: [ENTER ou mot de passe actuel]
Switch to unix_socket authentication? Y
Change the root password? Y
Remove anonymous users? Y
Disallow root login remotely? Y
Remove test database and access to it? Y
Reload privilege tables now? Y
```

CRÉATION D'UTILISATEURS ET PRIVILÈGES

CRÉATION D'UN NOUVEL UTILISATEUR

```
-- Connexion en tant que root
mariadb -u root -p

-- Créer un nouvel utilisateur
CREATE USER 'dev_user'@'localhost' IDENTIFIED BY 'mot_de_passe_securise';

-- Créer un utilisateur avec accès distant
CREATE USER 'dev_user'@'%' IDENTIFIED BY 'mot_de_passe_securise';
```

ATTRIBUTION DES PRIVILÈGES

```
GRANT [privilèges] ON [niveau_accès] TO [utilisateur];
```

```
-- Privilèges complets sur une base spécifique
GRANT ALL PRIVILEGES ON ma_base.* TO 'dev_user'@'localhost';

-- Privilèges limités
GRANT SELECT, INSERT, UPDATE ON ma_base.* TO 'dev_user'@'localhost';

-- Appliquer les changements
FLUSH PRIVILEGES;
```

Attribution des accès pour une base de données spécifiques

ATTRIBUTION DES PRIVILÈGES

La syntaxe avec le point (.) est essentielle car elle fait partie de la notation standard SQL pour désigner :

- La base de données(ma_base)
- La portée des objets dans cette base (* pour tous les objets)

Sans le point, la commande ne sera pas interprétée correctement et pourrait générer une erreur ou des privilèges incorrects.

ATTRIBUTION DES PRIVILÈGES

```
-- Correct : avec le point et l'astérisque
GRANT ALL PRIVILEGES ON ma_base.* TO 'dev_user'@'localhost';

-- Incorrect : sans le point
GRANT ALL PRIVILEGES ON ma_base TO 'dev_user'@'localhost';  -- NE PAS FAIRE
```

VÉRIFICATION DES PRIVILÈGES

```sql
-- Voir les privilèges d'un utilisateur
SHOW GRANTS FOR 'dev_user'@'localhost';

-- Voir tous les privilèges
SELECT * FROM information_schema.user_privileges;

-- Voir les privilèges sur les tables
SELECT * FROM information_schema.table_privileges;
```

VÉRIFICATION DES PRIVILÈGES

Quand vous créez ou mettez à jour des privilèges n'oubliez pas de les recharger pour qu'ils prennent effet.

```
FLUSH PRIVILEGES;
```

ATTRIBUTION DES PRIVILÈGES

```sql
-- Pour une table spécifique
GRANT ALL PRIVILEGES ON ma_base.ma_table TO 'dev_user'@'localhost';
```

Attribution des accès pour une table spécifique

ATTRIBUTION DES PRIVILÈGES

```
-- Pour toutes les bases de données
GRANT ALL PRIVILEGES ON *.* TO 'admin_user'@'localhost';
```

Attribution des accès pour toutes les bases de données

TYPES DE PRIVILÈGES DISPONIBLES

Privilèges de données

```sql
GRANT SELECT, INSERT, UPDATE, DELETE ON ma_base.* TO 'user'@'localhost';
```

TYPES DE PRIVILÈGES DISPONIBLES

Privilèges de structure

```sql
GRANT CREATE, DROP, ALTER ON ma_base.* TO 'admin'@'localhost';
```

TYPES DE PRIVILÈGES DISPONIBLES

Privilèges administratifs

```
GRANT CREATE USER, RELOAD, PROCESS ON *.* TO 'super_admin'@'localhost';
```

RETIRER DES PRIVILÈGES

```sql
-- Retirer des privilèges spécifiques
REVOKE INSERT, UPDATE ON ma_base.* FROM 'user'@'localhost';

-- Retirer tous les privilèges
REVOKE ALL PRIVILEGES ON ma_base.* FROM 'user'@'localhost';
```

PREMIER CONTACT AVEC MARIADB

INTERFACE DE COMMANDE

Connexion à MariaDB

```
mariadb -u root -p
```

INTERFACE DE COMMANDE

Commandes de bases CLI

```sql
-- Voir les bases de données existantes
SHOW DATABASES;

-- Voir la version de MariaDB
SELECT VERSION();

-- Voir l'utilisateur actuel
SELECT CURRENT_USER();

-- Voir les variables système
SHOW VARIABLES;
```

INTERFACE GRAPHIQUES RECOMMANDÉES

phpMyAdmin :

- Interface web
- Facile à utiliser
- Très populaire

INTERFACE GRAPHIQUES RECOMMANDÉES

DBeaver :

- Multi-plateforme (Java)
- Support de nombreux SGBD
- Fonctionnalités avancées

CRÉATION D'UNE PREMIÈRE BASE

```
CREATE DATABASE IF NOT EXISTS bibliotheque
CHARACTER SET utf8mb4
COLLATE utf8mb4_unicode_ci;
```

- CREATE DATABASE: Crée une nouvelle base de données
- IF NOT EXISTS: Évite une erreur si la base existe déjà
- CHARACTER SET utf8mb4: Support complet Unicode, incluant les émojis
- COLLATE utf8mb4_unicode_ci: Définit les règles de tri (insensible à la casse)

SÉLECTION DE LA BASE

```
USE bibliotheque;
```

- USE: Indique quelle base de données sera utilisée

MODÉLISATION

MODÈLE ENTITÉ-ASSOCIATION

DÉFINITION DU MODÈLE EA

- Le modèle EA est conceptuel, il représente les données de façon abstraite et proche du monde réel.
- Ce modèle utilise des associations avec des cardinalités (1-1, 1-N, N-M)

ENTITÉS

Définition: Objet concret ou abstrait du monde réel

Caractéristiques:

- Identifiable de manière unique
- Possède des attributs
- A une existence propre

ENTITÉS

Exemple :

```
ENTITÉ: ÉTUDIANT
Attributs:
- Numéro étudiant (identifiant)
- Nom
- Prénom
- Date de naissance
- Email
```

ATTRIBUTS

Types d'attributs:

- Simple (nom, prénom)
- Composé (adresse = rue + ville + code postal)
- Monovalué (date de naissance)
- Multivalué (téléphones)
- Dérivé (âge calculé à partir de la date de naissance)

ATTRIBUTS

Identifiants:

- Naturel (numéro de sécurité sociale)
- Artificiel (ID auto-incrémenté)

RELATIONS

Définition: Association entre entités

Caractéristiques:

- Nom
- Cardinalités

RELATIONS

Exemple de relations:

```
EMPRUNTE (ÉTUDIANT - LIVRE)
- Date d'emprunt
- Date de retour prévue
```

CARDINALITÉS

Format: (min, max)

Valeurs possibles:

- 0,1 : optionnel, unique
- 1,1 : obligatoire, unique
- 0,n : optionnel, multiple
- 1,n : obligatoire, multiple

EXEMPLE D'UN MODÈLE EA

Exemple:

```
ÉTUDIANT (1,n) ----EMPRUNTE---- (0,n) LIVRE
Signification:
- Un étudiant doit emprunter au moins un livre
- Un livre peut être emprunté par plusieurs étudiants ou aucun
```

SCHÉMAS RELATIONNELS

DÉFINITION DU SCHÉMA RELATIONNEL

- Le schéma relationnel est logique, plus technique et orienté vers l'implémentation en base de données.
- Ce modèle utilise des clés étrangères et des tables de jointure

OUTILS DE MODÉLISATION

- Mermaid
- LucidChart
- Draw.io

TYPES DE RELATIONS

Relation 1:1

```
erDiagram
    EMPLOYE ||--|| BADGE : possède

    EMPLOYE {
        int id
        string nom
        string prenom
    }

    BADGE {
        int id
        string numero
        date date_creation
    }
```

TYPES DE RELATIONS

Relation 1:N

```
erDiagram
    DEPARTEMENT ||--o{ EMPLOYE : contient

    DEPARTEMENT {
        int id
        string nom
        string localisation
    }

    EMPLOYE {
        int id
        string nom
        string prenom
        int departement_id
    }
```

TYPES DE RELATIONS

Relation N:M

Dans le cas d’une relation Many-To-Many, il n’est pas possible de lier deux tables directement, il faut utiliser ce qu’on appelle une table de jonction.

❌ Pourquoi c'est impossible ?

- On ne peut pas stocker plusieurs valeurs dans un champ
- On ne sait pas où mettre la référence à l'autre table
- Impossible de stocker des informations supplémentaires sur la relation

TYPES DE RELATIONS

❌

Relation
N:M direct

```
erDiagram
    ETUDIANT }|--|{ COURS : suit

    ETUDIANT {
        int id PK
        string nom
        string prenom
    }

    COURS {
        int id PK
        string nom
        int credits
    }
```

TYPES DE RELATIONS

✅

Relation N:M avec table de jonction

```
erDiagram
    ETUDIANT ||--o{ INSCRIPTION : "participe à"
    INSCRIPTION }o--|| COURS : "concerne"

    ETUDIANT {
        int id PK
        string nom
        string prenom
    }

    INSCRIPTION {
        int etudiant_id FK
        int cours_id FK
        date date_inscription
        decimal note
    }

    COURS {
        int id PK
        string nom
        int credits
    }
```

CLÉS

- Primaires: Identifiant unique
- Étrangères: Référence à une autre table
- Candidates: Alternatives possibles pour clé primaire

EXERCICE MODÉLISATION

EXERCICE

Créez à présent votre première modélisation.

Nous allons utiliser pour cet exercice une table CATEGORIES et une tables LIVRES.

Vous êtes libres de choisir les attributs de chaque table et de les typer de façon cohérente.

EXERCICE - CORRIGÉ

```
erDiagram
    CATEGORIES ||--o{ LIVRES : contient

    CATEGORIES {
        int id PK "Auto-increment"
        string nom "VARCHAR(50) NOT NULL"
        text description "Nullable"
        timestamp created_at "DEFAULT CURRENT_TIMESTAMP"
        timestamp updated_at "Auto update"
    }

    LIVRES {
        int id PK "Auto-increment"
        string isbn "VARCHAR(13) UNIQUE"
        string titre "VARCHAR(100) NOT NULL"
        string auteur "VARCHAR(100) NOT NULL"
        string editeur "VARCHAR(50) Nullable"
        year annee_publication "Nullable"
        int categorie_id FK "Références CATEGORIES"
        int nb_exemplaires "DEFAULT 1"
        timestamp created_at "DEFAULT CURRENT_TIMESTAMP"
    }
```

MODÉLISATION

Relations:

- La flèche ||--o{ indique une relation "un-à-plusieurs"
- Une catégorie peut avoir plusieurs livres (o{)
- Un livre appartient à une seule catégorie (||)

Table CATEGORIES:

- PK = Primary Key (Clé primaire)
- Les champs obligatoires sont notés "NOT NULL"
- Les timestamps sont gérés automatiquement

Table LIVRES:

- FK = Foreign Key (Clé étrangère)
- UNIQUE sur ISBN garantit l'unicité
- Valeurs par défaut indiquées
- Relations établies via categorie_id

Structure logique:

- Une catégorie peut contenir plusieurs livres
- Un livre appartient à une seule catégorie
- Les dates sont gérées automatiquement
- Intégrité référentielle assurée par la clé étrangère

CONCEPTS AVANCÉS

CONTRAINTES D'INTÉGRITÉ

```sql
-- Contrainte de domaine
age INT CHECK (age >= 0 AND age < 150)

-- Contrainte d'unicité
email VARCHAR(100) UNIQUE

-- Contrainte de non-nullité
nom VARCHAR(50) NOT NULL

-- Contrainte référentielle
FOREIGN KEY (departement_id) REFERENCES departements(id)
    ON DELETE CASCADE
    ON UPDATE CASCADE
```

CONTRAINTES DE DOMAINES

Les contraintes de domaine permettent de définir les valeurs acceptables pour une colonne.

CONTRAINTES DE DOMAINES

```sql
-- Structure de base
CREATE TABLE employes (
    id INT PRIMARY KEY,
    age INT CHECK (age >= 0 AND age < 150),
    salaire DECIMAL(10,2) CHECK (salaire >= 0),
    niveau VARCHAR(10) CHECK (niveau IN ('Junior', 'Senior', 'Expert'))
);
```

```sql
-- Insertion valide
INSERT INTO employes (id, age, salaire, niveau)
VALUES (1, 25, 3000, 'Junior'); -- OK

-- Insertions invalides
INSERT INTO employes (id, age, salaire, niveau)
VALUES (2, -5, 3000, 'Junior');  -- Erreur: age négatif
INSERT INTO employes (id, age, salaire, niveau)
VALUES (3, 25, -1000, 'Stagiaire');  -- Erreur: salaire négatif et niveau invalide
```

CONTRAINTES D'UNICITÉ

Les contraintes de domaine permettent de définir les valeurs acceptables pour une colonne.

```sql
CREATE TABLE utilisateurs (
    id INT PRIMARY KEY,
    email VARCHAR(100) UNIQUE,
    numero_secu VARCHAR(15) UNIQUE
);

-- Peut aussi être défini pour plusieurs colonnes
CREATE TABLE reservations (
    id INT PRIMARY KEY,
    salle_id INT,
    date_reservation DATE,
    UNIQUE(salle_id, date_reservation)  -- Combinaison unique
);
```

CONTRAINTES DE NON-NULLITÉ

Empêche l'insertion de valeurs NULL dans une colonne.

```sql
CREATE TABLE clients (
    id INT PRIMARY KEY,
    nom VARCHAR(50) NOT NULL,
    prenom VARCHAR(50) NOT NULL,
    adresse VARCHAR(200),  -- Peut être NULL
    telephone VARCHAR(15) NOT NULL
);
```

CONTRAINTES RÉFÉRENTIELLES

Établit et maintient une relation entre les données de deux tables.

ON DELETE:

- CASCADE: Supprime les enregistrements liés
- SET NULL: Met NULL dans les enregistrements liés
- RESTRICT: Empêche la suppression (par défaut)
- NO ACTION: Similaire à RESTRICT

ON UPDATE:

- CASCADE: Propage les modifications
- SET NULL: Met NULL dans les références
- RESTRICT: Empêche la modification (par défaut)
- NO ACTION: Similaire à RESTRICT

CONTRAINTES RÉFÉRENTIELLES

```sql
-- Tables avec différentes contraintes référentielles
CREATE TABLE departements (
    id INT PRIMARY KEY,
    nom VARCHAR(50) NOT NULL UNIQUE
);

CREATE TABLE employes (
    id INT PRIMARY KEY,
    nom VARCHAR(50) NOT NULL,
    departement_id INT,
    superviseur_id INT,
    FOREIGN KEY (departement_id)
        REFERENCES departements(id)
        ON DELETE CASCADE  -- Si département supprimé, supprime les employés
        ON UPDATE CASCADE, -- Si ID département modifié, met à jour les références
    FOREIGN KEY (superviseur_id)
        REFERENCES employes(id)
        ON DELETE SET NULL  -- Si superviseur supprimé, met NULL
        ON UPDATE CASCADE
);
```

CONTRAINTES RÉFÉRENTIELLES

```sql
-- Insertion des données
INSERT INTO departements VALUES (1, 'Informatique');
INSERT INTO departements VALUES (2, 'Comptabilité');

-- Insertion d'employés
INSERT INTO employes (id, nom, departement_id)
VALUES (1, 'Jean', 1);  -- OK

-- Tentatives avec violations de contraintes
INSERT INTO employes (id, nom, departement_id)
VALUES (2, 'Marie', 3);  -- Erreur: département inexistant

-- Test des actions ON DELETE
DELETE FROM departements WHERE id = 1;
-- Avec CASCADE: Supprime aussi tous les employés du département
-- Avec SET NULL: Met departement_id à NULL pour ces employés
-- Avec RESTRICT: Empêche la suppression du département

-- Test des actions ON UPDATE
UPDATE departements SET id = 10 WHERE id = 2;
-- Avec CASCADE: Met à jour departement_id des employés
-- Avec SET NULL: Met departement_id à NULL
-- Avec RESTRICT: Empêche la modification
```

EXERCICE PRATIQUE DE MODÉLISATION

SYSTÈME DE GESTION D'UNE ÉCOLE DE MUSIQUE

https://drive.google.com/file/d/1MJLbR2KZv4ysNWDqFb55UxKQyOdxKAyV/view?usp=drive_link

CONCEPTIONS DE SCHÉMAS RELATIONNELS

NORMALISATION

INTRODUCTION AUX FORMES NORMALES

Les formes normales permettent d'éviter les anomalies de données et la redondance.

- Exemple non normalisé

```
erDiagram
    %% Non Normalisé - Problème de structure
    FACTURE_NON_NORMALISEE {
        int id
        string client_nom
        string client_adresse
        string produit1_nom
        decimal produit1_prix
        int produit1_quantite
        string produit2_nom
        decimal produit2_prix
        int produit2_quantite
        decimal total
    }
```

PREMIÈRE FORME NORMALE (1NF)

- Chaque colonne contient une valeur atomique
- Pas de répétition de groupes de colonnes

EXEMPLE 1NF

```
erDiagram
    %% 1NF - Atomicité des données
    FACTURE ||--|{ LIGNE_FACTURE : contient

    FACTURE {
        int id PK
        string client_nom
        string client_adresse
        decimal total
    }

    LIGNE_FACTURE {
        int facture_id FK
        string produit_nom
        decimal prix
        int quantite
    }
```

DEUXIÈME FORME NORMALE (2NF)

- Doit être en 1NF
- Tous les attributs non-clés dépendent de toute la clé primaire

EXEMPLE 2NF

```
%% 2NF - Dépendance complète de la clé
FACTURE ||--|{ LIGNE_FACTURE : contient
PRODUIT ||--o{ LIGNE_FACTURE : compose
CLIENT ||--o{ FACTURE : possède

CLIENT {
    int id PK
    string nom
    string adresse
}

FACTURE {
    int id PK
    int client_id FK
    date date_facture
    decimal total
}

PRODUIT {
    int id PK
    string nom
    decimal prix
}

LIGNE_FACTURE {
    int facture_id FK
    int produit_id FK
    int quantite
}
```

TROISIÈME FORME NORMALE (3NF)

- Doit être en 2NF
- Pas de dépendances transitives

EXEMPLE 3NF

```
erDiagram
    %% 3NF - Élimination des dépendances transitives
    CLIENT ||--o{ FACTURE : possède
    VILLE ||--o{ CLIENT : habite
    FACTURE ||--|{ LIGNE_FACTURE : contient
    PRODUIT ||--o{ LIGNE_FACTURE : compose
    CATEGORIE ||--o{ PRODUIT : classifie

    VILLE {
        int id PK
        string nom
        string code_postal
        string pays
    }

    CLIENT {
        int id PK
        string nom
        int ville_id FK
    }

    FACTURE {
        int id PK
        int client_id FK
        date date_facture
    }

    PRODUIT {
        int id PK
        string nom
        decimal prix
        int categorie_id FK
    }

    CATEGORIE {
        int id PK
        string nom
        string description
    }

    LIGNE_FACTURE {
        int facture_id FK
        int produit_id FK
        int quantite
    }
```

TP
SCHÉMA DU CMS

CONSIGNES

https://drive.google.com/file/d/142eP6Z2s_xboFHMkgXrtYCRljWOOuSF3/view?usp=drive_link