Lycée TOUCHARD WASHINGTON

Développement Logiciel

Merise

Table des matières

1. Introduction

La méthode **Merise**, développée en France dans les années 1970 par la société **Cegedim**, est une approche structurée de modélisation et de conception des systèmes d'information. Principalement utilisée dans le monde francophone, elle se concentre sur la conception de bases de données et les études associées aux systèmes de gestion.

Au fur et à mesure des évolutions technologiques et des nouvelles pratiques de développement, Merise a été améliorée et adaptée sous le nom de **Merise 2**. Cette nouvelle version apporte plusieurs évolutions importantes :

- **Extension de la modélisation fonctionnelle** : nouvelle approche de la modélisation des traitements, en intégrant des concepts modernes de l'architecture des systèmes d'information, tels que les traitements par événements ou les systèmes répartis.
- **Prise en compte des objets** : adaptation aux concepts pour intégrer la modélisation orientée objet. Cela permet de modéliser les entités sous forme d'objets avec leurs attributs et leurs méthodes, et de mieux répondre aux besoins des applications orientées objet.
- **Modélisation de l'architecture distribuée** : Accent sur la modélisation des systèmes répartis, avec des outils pour représenter la communication entre les différentes parties du système (serveur, client, etc.), essentiel dans les architectures modernes basées sur les réseaux et le cloud.

Merise 2 introduit différents quatre niveaux d’abstraction qui sont décrits à la suite.

2. Les différents niveaux d'abstraction de Merise

- **Le niveau organisationnel : Quoi ?**

  Ce niveau décrit le système dans son **environnement métier et humain**, sans prendre en compte l’aspect informatique. Il analyse les **flux d’information** et les **processus de l’entreprise**.

  L’objectif principal est :

  - **d’identifier les acteurs** (utilisateurs, unités organisationnelles, partenaires),
  - de comprendre les **processus métiers** et leurs interactions,
  - et de repérer les **informations** échangées entre les acteurs.

  Exemple : **Gestion d’une bibliothèque**

  Règles de gestion :

  - *Le système doit permettre aux lecteurs d’emprunter des livres.*
  - *Il s’assure que le nombre de livres empruntés par lecteur reste limité.*

  Ces règles traduisent les objectifs métier principaux : ***faciliter l’accès aux ressources tout en garantissant une gestion efficace du stock.***

- **Le niveau conceptuel : D'où ? Qui ? Quand ?**

  À ce niveau, le système est décrit **sans contrainte technique ou organisationnelle**. Il formalise les **données** et les **traitements** de manière abstraite et indépendante des outils ou de la technologie.

  Exemple :

  Règles d'organisation :

  - *Les lecteurs peuvent réserver ou emprunter un ou plusieurs livres.*
  - *Un livre emprunté doit être retourné avant d'être emprunté par un autre lecteur.*

  Ces règles établissent les bases de l'organisation du système.

- **Le niveau logique : Quand ? Où ? Comment ?**

  Ce niveau traduit les modèles conceptuels en modèles **compatibles avec les technologies** choisies (bases de données, protocoles, etc.), mais sans tenir compte des contraintes physiques comme le stockage ou les performances.

  Exemple :

  Règles techniques :

  - *Les emprunts sont gérés par un système informatique. Chaque livre et chaque lecteur sont identifiés dans une base de données par un **identifiant unique**.*
  - *Les réservations et emprunts sont validés à l'aide de requêtes **SQL**.*

- **Le niveau physique : Comment ?**

  Ce dernier niveau concerne l'implémentation concrète du système, en tenant compte des contraintes matérielles et logicielles (SGBD, matériel, réseau, performances).

  Exemple :

  Règles d'implémentation :

  - *Les informations sur les livres et les lecteurs sont stockées dans une base MySQL.*
  - *Les requêtes pour emprunter ou retourner un livre sont exécutées via des **scripts PHP** interagissant avec la base de données.*
  - *Les notifications de retour sont envoyées par **e-mail** en utilisant un serveur SMTP.*

Les deux premiers niveaux (**organisationnel** et **conceptuel**) sont totalement **indépendants de la technologie**. Ils se concentrent sur l'analyse métier et la modélisation abstraite. À partir du niveau logique, des choix techniques commencent à apparaître, jusqu'au niveau physique qui est entièrement **dépendant des outils** choisis.

3. Détails du niveau organisationnel dans Merise

Le niveau organisationnel se concentre sur la compréhension du système dans son **environnement métier et humain**. Il s'agit d'une étape essentielle qui prépare la modélisation en identifiant les besoins, les acteurs et les flux d'information.

3.1. Objectifs du niveau organisationnel

- **Comprendre les processus métiers** : Quelles sont les actions réalisées par les acteurs ?
- **Identifier les flux d'information** : Quelles données circulent et entre qui ?
- **Clarifier les rôles et responsabilités** : Qui fait quoi dans l'organisation ?
- **Définir les objectifs principaux** du système.

3.2. Outils et moyens à mettre en œuvre

3.2.1. Analyse des processus métiers

Pour identifier et décrire les activités principales :

Outil : Cartographie des processus

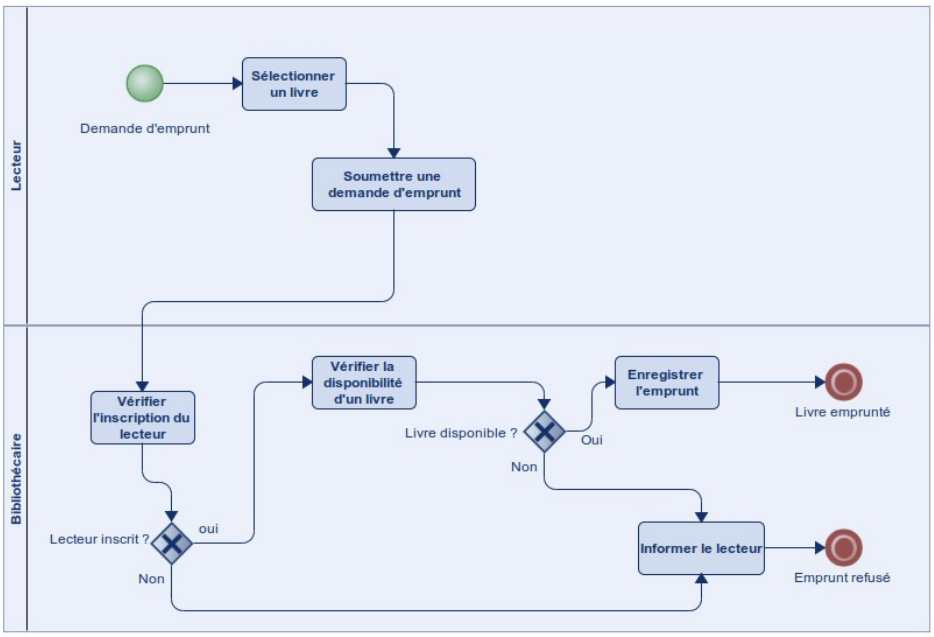
- Représente visuellement les activités et leur enchaînement logique.
- Chaque processus peut être détaillé à l'aide d'un diagramme, tel qu'un **diagramme BPMN** (**Business Process Model and Notation**), qui est une norme graphique internationale.
  - **BPMN** permet de modéliser les étapes, les flux d'informations et les interactions des processus métiers de manière claire et compréhensible.
  - Les principaux éléments d'un diagramme BPMN incluent les **activités** (tâches ou sous-processus), les **événements** (début, intermédiaire, fin), les **passerelles** (décisions), les **flux** (séquence ou message), et les **participants** (piscines ou pools et couloirs ou lanes).
  - Cette approche favorise une compréhension partagée entre les experts métiers et les techniciens, tout en facilitant l'identification d'améliorations ou l'automatisation des processus.

Moyens :

- Interviews avec les acteurs clés.
- Analyse documentaire (procédures internes, organigrammes, etc.).

Exemple : Pour la suite du document, les illustrations porteront sur la gestion d'emprunt de livres dans une bibliothèque évoquée précédemment.

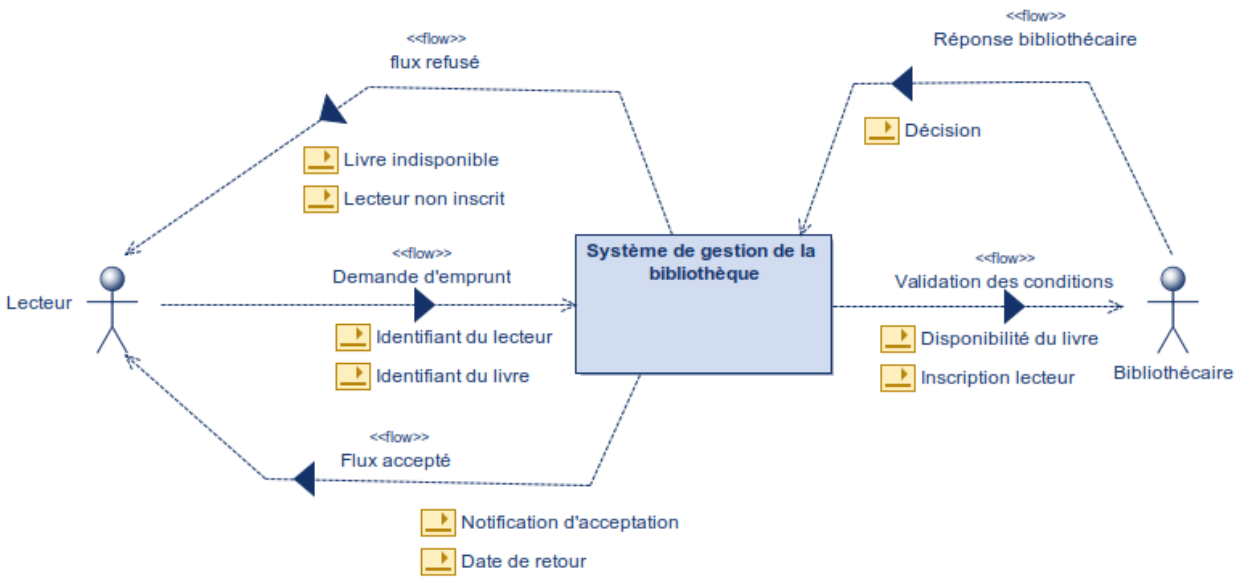
Exemple : Gérer les emprunts **BPMN diagramme de processus**



3.2.2. Analyse des flux d'information

Pour identifier les **informations échangées** entre acteurs, un **diagramme de flux de données** (DFD) peut être utilisé. Celui-ci représente les flux d'information entre les processus et les acteurs et montre **d'où vient** et **où va** chaque information.

Exemple : Gérer les emprunts **Diagramme de flux de données**



Ce diagramme est réalisé sous **Modelio** à partir d'un diagramme de classes.

3.2.3. Identification des acteurs et des responsabilités

Pour comprendre **qui intervient dans chaque processus,** une matrice de type RACI (Responsable, Autorité, Consulté, Informé) permet d'associer les rôles aux processus pour clarifier les responsabilités et de définir les interactions et dépendances organisationnelles.

Processus	Lecteur	Bibliothécaire	Système
Faire une demande	R	A	-
Vérifier disponibilité	C	R	A
Enregistrer l'emprunt	I	R	A
Informer le lecteur	A	-	R

- **R** : Responsable (effectue l'action).
- **A** : Autorité (prend les décisions finales).
- **C** : Consulté (donne un avis ou fournit des informations).
- **I** : Informé (est tenu au courant des résultats).

3.2.4. Définition des règles de gestion

les **règles métier** sont essentielles au système pour décrire les **contraintes** qui régissent les processus et identifier les **exceptions** et cas particuliers.

Exemple de règles pour l'emprunt :

- Un lecteur peut emprunter **jusqu'à 3 livres simultanément**.
- Les emprunts sont autorisés uniquement si **le lecteur est inscrit**.
- La durée maximale d'emprunt est de **30 jours**.

4. Détail du niveau conceptuel

Le niveau conceptuel est un approfondissement essentiel qui structure les données et les interactions identifiées au niveau organisationnel. Il sert de base solide pour les étapes techniques (logique et physique) et garantit que toutes les données et règles de gestion sont correctement définies. Les aspects techniques de mise en œuvre ne sont pas encore pris en compte à ce niveau.

4.1. Objectifs du niveau conceptuel

D'écrire de manière structurée :

- **Les entités clés,** acteurs et systèmes impliqués.
- **Les informations échangées** sous forme de données structurées.
- **Les relations** entre ces données, leurs traitements, et leur flux dans le processus.

Dans le cadre de notre exemple de gestion d'emprunt de livre, les entités clés identifiées sont au nombre de trois :

Entité	Description
Lecteur	• Soumet une demande d'emprunt. • Reçoit une réponse (acceptation ou refus).
Bibliothécaire	• Vérifie les conditions (inscription du lecteur, disponibilité du livre). • Valide et enregistre l'emprunt ou refuse la demande.
Système de gestion de la bibliothèque	• Gère les données (lecteurs, livres, emprunts). • Joue un rôle d'intermédiaire entre le lecteur et le bibliothécaire.

Les informations échangées sont représentées dans un diagramme entités-relations, qui structure et détaille le contenu des données.

Exemple :

Dans le cas présenté, un lecteur peut **emprunter** plusieurs livres, et un livre peut être emprunté par plusieurs lecteurs. Cette situation illustre une relation de type N-N, indiquant que plusieurs occurrences d'une entité (Lecteurs) peuvent être associées à plusieurs occurrences d'une autre entité (Livres).

Ces cardinalités, qui définissent le nombre d'occurrences impliquées dans chaque relation, jouent un rôle clé dans la modélisation des interactions entre entités et seront détaillées dans le prochain chapitre.

4.2. Cardinalités des relations

Les cardinalités décrivent le nombre de relations possibles entre les occurrences de deux entités dans un modèle de données. Elles permettent de structurer les interactions entre les données. Voici une description détaillée des principaux types de cardinalité :

4.2.1. Relation 1-1 (un à un)

- **Description :**

  Une occurrence de l'entité A est associée à une seule occurrence de l'entité B, et réciproquement.

- **Exemples :**

  Lecteurs et Cartes : Un lecteur de la bibliothèque possède une et une seule carte d'adhérent. Une carte d'adhérent est associée à un seul lecteur.

4.2.2. Relation 1-N (Un à plusieurs)

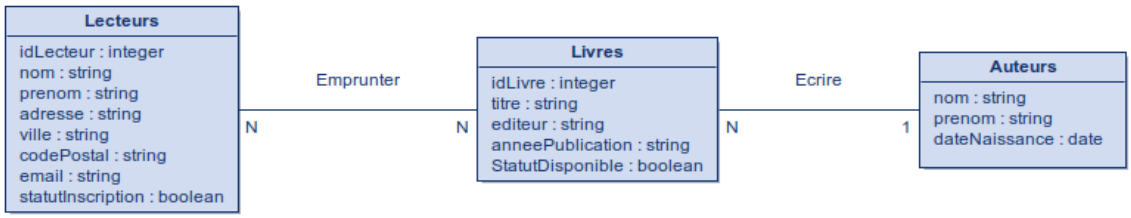
- **Description :**

  Une occurrence de l'entité A peut être liée à plusieurs occurrences de l'entité B. En revanche, une occurrence de l'entité B est liée à une seule occurrence de l'entité A.

- **Exemples :**

  Auteurs et ***Livres*** : Un auteur peut écrire plusieurs livres, mais chaque livre est écrit par un seul auteur.

Exemple :



Dans cette représentation plus complète, le nom de l'auteur est retiré de ***Livres***, la relation ***Ecrire*** relie ***Livres*** à ***Auteurs*** qui contient en plus le prénom et la date de naissance de l'auteur pour enrichir l'information en évitant la redondance des informations sur l'auteur dans chaque livre.

4.2.3. Relations 0-1 et 0-N

Certaines relations sont optionnelles, ce qui signifie qu'elles ne s'appliquent pas systématiquement à toutes les occurrences des entités impliquées. Ces relations peuvent être de type 0-1 ou 0-N.

Exemples :
Une relation de type 0-1 s'applique lorsqu'un lecteur nouvellement enregistré peut ne pas encore avoir d'emprunt en cours.

Une relation de type 0-N peut illustrer un système où certains lecteurs n'ont pas encore emprunté de livres, tandis que d'autres peuvent en avoir plusieurs.

4.2.4. Relation N-N (Plusieurs-à-plusieurs)

- **Description :**

  Une occurrence de l'entité A peut être liée à plusieurs occurrences de l'entité B.

  Une occurrence de l'entité B peut être liée à plusieurs occurrences de l'entité A.

- **Exemples :**

  C'est le cas de notre exemple ou un lecteur peut emprunter plusieurs livres et un livre peut être emprunté par plusieurs lecteurs.

- **Transformation des relations N-N**

  Toutes les relations de type N-N doivent être ramenées à une relation de type 1-N en introduisant une table intermédiaire en vue d'une implémentation correcte dans un système de gestion de base de données relationnelle.

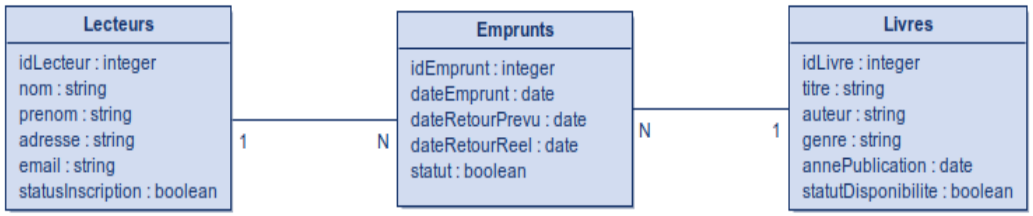
Dans notre exemple, la relation **Emprunter** entre **Lecteurs** et **Livres** est transformée en introduisant une table intermédiaire appelée **Emprunts**. Cette table contient des attributs spécifiques à cette relation :

- **date d'emprunt**,
- **date de retour prévu**,
- **date de retour réel**.
- **statut**

Cette transformation crée deux nouvelles relations :

- Une relation de cardinalité **(1,N)** entre **Lecteurs** et **Emprunts**.
- Une relation de cardinalité **(N,1)** entre **Emprunts** et **Livres**.

Exemple : **Transformation de la relation entre Lecteurs et Livres**



4.3. Outils de conceptualisation de la base de données

Pour élaborer le Modèle Conceptuel de Données (MCD) ou modèle entité-relation outre Modelio avec le diagramme de classes, il existe des outils en ligne comme Mocodo qui permettent de réaliser un modèle conceptuel de données décrit dans un langage textuel minimaliste et d'obtenir dans un premier temps une représentation graphique de ce modèle : https://www.mocodo.net/

La définition des entités sous le schéma obtenu montre que les clés primaires des entités ***Lecteurs*** et ***Livres*** sont également devenues des clés étrangères dans la table **Emprunts**. Cela permet de matérialiser la relation ***Emprunte***r en assurant le lien entre chaque emprunt et les entités ***Lecteurs*** et ***Livres*** correspondantes.

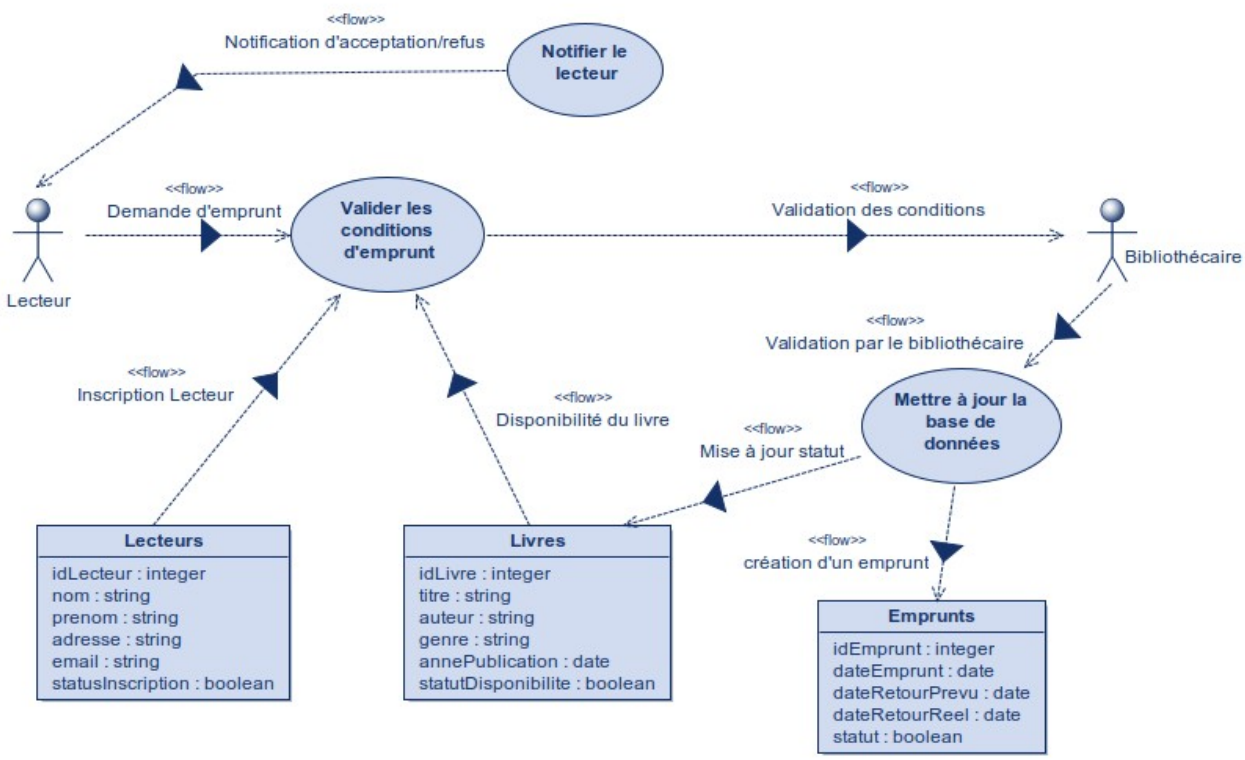
Remarque :

Dans un premier temps l'outil recommande l'usage des espaces, accents et signes de ponctuation corrects pour une meilleure lisibilité. Le premier attribut d'une identité est considéré comme un identifiant, il est souligné. Pour souligner d'autres attributs, il est nécessaire de les préfixer par un tiret bas.

4.4. Enrichissement du diagramme de flux de données

Au niveau conceptuel, le digramme de flux de données fait apparaître de nouvelles informations comme les processus et les tables de la base de données.

Exemple : **Enrichissement du diagramme de flux de données**



5. Détail du niveau logique de données

Le niveau logique est une étape intermédiaire entre la modélisation conceptuelle et la mise en œuvre physique des données. Il consiste à traduire le **Modèle Conceptuel de Données (MCD)** en un **Modèle Logique de Données (MLD)**, tout en prenant en compte les contraintes spécifiques des systèmes de gestion de bases de données relationnelles (SGBDR).

5.1. Objectifs du Niveau Logique

1. **Traduire le MCD en MLD :**
   - Adapter le modèle conceptuel pour le rendre compatible avec un SGBDR.
   - Transformer les entités, les relations, et les cardinalités en structures relationnelles.
2. **Garantir la cohérence des données :**
   - Préserver les dépendances fonctionnelles identifiées au niveau conceptuel.
   - Assurer l'intégrité des relations entre les données.
3. **Préparer la mise en œuvre physique :**
   - Identifier les clés primaires et étrangères.
   - Introduire des tables d'association pour les relations N−N.
   - Normaliser les relations pour réduire les redondances et éviter les anomalies.

5.2. Transition du MCD vers le MLD

La transition du MCD au MLD suit des règles bien définies :

5.2.1. Transformation des entités

- Chaque **entité forte** du MCD devient une **table relationnelle** dans le MLD.
- Les attributs de l'entité deviennent les colonnes de la table.
- La clé primaire identifiée au niveau conceptuel est conservée.

5.2.2. Transformation des associations

1. **Relations 1−1 :**
   - Intégrer la clé primaire d'une entité comme clé étrangère dans l'autre entité.
   - Si les deux entités ont une dépendance totale, elles peuvent être fusionnées en une seule table.
2. **Relations 1−N :**
   - Ajouter la clé primaire de l'entité 1 comme clé étrangère dans l'entité N.
3. **Relations N−N :**
   - La relation doit être implémentée sous la forme d'une table d'association. Cette table contient, les clés primaires des deux entités comme clés étrangères et des attributs supplémentaires si nécessaire par exemple la date d'emprunt celle de retour prévu…
   - Les deux clés étrangères éventuellement combinées avec un autre attribut (ici date d'emprunt) peuvent devenir la clé primaire de la table ou un identifiant unique peut être cette clé primaire.

5.2.3. Exemple pratique complet :

MCD Initial : Les entités et relations suivantes sont identifiées :

MCD avec la relation N-N

Entités :

1. **Lecteurs** :
   - Attributs : `idLecteur (identifiant unique), nom, prenom, dateNaissance, adresse, email, statutInscription.`
2. **Livres** :
   - Attributs : `idLivre (identifiant unique), titre, auteur, genre, anneePublication, statutDisponibilite.`

Relation :

1. Un lecteur peut emprunter plusieurs livres (N−N).
2. Un livre peut être emprunté par plusieurs lecteurs (N−N).

Création de la table intermédiaire

Emprunts (relation N-N entre Lecteurs et Livres) :

- Attributs spécifiques à la relation :
  - `dateEmprunt`
  - `dateRetourPrevu`
  - `dateRetourReel`
  - `statut`

Les attributs spécifiques à l'emprunt (**dateEmprunt**, **dateRetourPrevu**, **dateRetourReel, statut**) sont portés par la relation **Emprunts**, car ils sont propres à chaque association entre un lecteur et un livre.

- Multiplicité :
  - 1,N côté **Lecteurs** (un lecteur peut emprunter plusieurs livres).
  - 1,N côté **Livres** (un livre peut être emprunté plusieurs fois par différents lecteurs à des moments différents).

Transformation en MLD

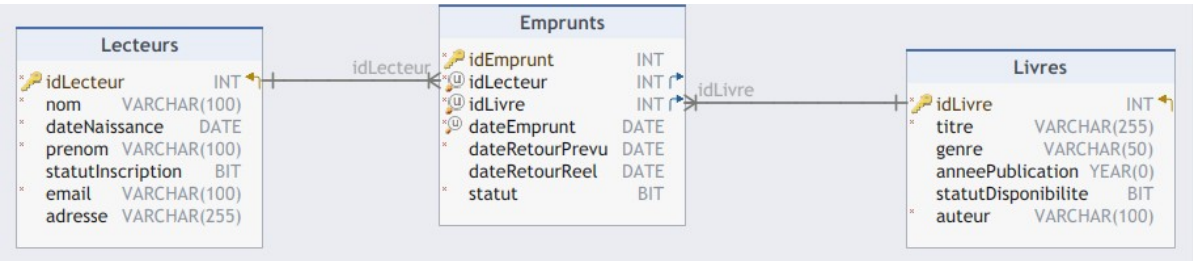
1. **Entités transformées en tables :**
   - Table **Lecteurs**:
     - ***Colonnes*** : idLecteur (PK), nom, prenom, dateNaissance, adresse, email, statutInscription
   - Table **Livres** :
     - ***Colonnes*** : idLivre (PK), titre, auteur, genre, anneePublication, statutDisponibilité
2. **Table d'association pour N−N :**
   - Table **Emprunts** :
     - ***Colonnes*** : idEmprunt (PK), idLecteur (FK), idLivre (FK), dateEmprunt, dateRetourPrevu, dateRetourReel, statut

5.3. Outils et Moyens à Mettre en Œuvre

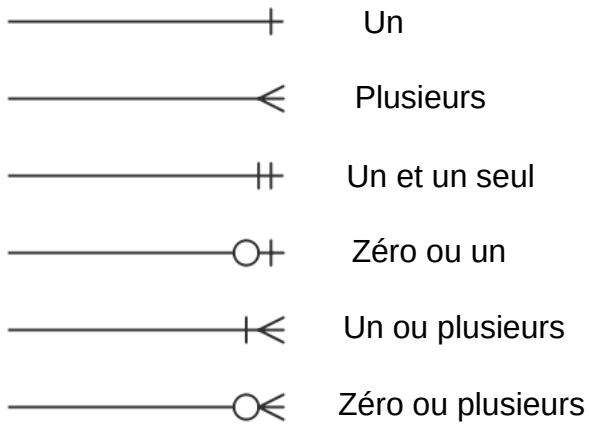
5.3.1. Modélisateurs de bases de données :

- **Mocodo** peut être utilisé pour générer les MLD automatiquement à partir des MCD et ainsi obtenir un script SQL.
- Des logiciels comme **BDSchema, DB Browser for SQLite** ou d'autres permettent également une représentation graphique. L'outil est plus sophistiqué du point de vue utilisation.
- Outils de vérification : Des scripts SQL peuvent être utilisés pour tester la conformité des tables (exemple : vérification des contraintes d'intégrité).

Modèle obtenu avec DBSchema



Légende pour la cardinalité :



5.3.2. Moyens

1. **Analyse des besoins :**
   - Réaliser des ateliers collaboratifs pour valider les entités et les relations.
   - Revoir les dépendances fonctionnelles identifiées au MCD.
2. **Documentation technique :**
   - Créer un document décrivant chaque relation et ses contraintes.
   - Fournir une matrice d'attribution des clés primaires et étrangères.
3. **Tests de cohérence :**
   - Tester la structure logique avec des cas d'usage.
   - Simuler des requêtes SQL sur des jeux de données fictifs.

6. Détail du niveau physique

Le niveau physique consiste à traduire le MLD en une structure physique adaptée aux spécificités techniques du SGBD choisi. Cela inclut la définition des types de données, des contraintes, des index, et des mécanismes d'optimisation pour répondre aux besoins de performance et d'intégrité des données. Les exemples du niveau physique utiliseront Mysql.

6.1. Objectifs du niveau physique

- **Optimisation des performances :**
  - Minimiser le temps d'exécution des requêtes grâce à des index et des choix de types de données adaptés.
  - Gérer les grands volumes de données en utilisant des mécanismes tels que la partition de tables ou le stockage en colonnes.
- **Intégrité des données :**
  - Implémenter des contraintes (`PRIMARY KEY`, `FOREIGN KEY`, `UNIQUE`, `NOT NULL`, etc.) pour assurer la cohérence et l'exactitude des données.
- **Adaptation au SGBD choisi :**
  - Tirer parti des spécificités du SGBD, comme les types de données propres au système (`AUTO_INCREMENT`).
- **Gestion de la concurrence :**
  - Définir des stratégies de verrouillage et d'accès concurrentiel pour éviter les conflits entre utilisateurs simultanés.

6.2. Éléments caractéristiques du niveau physique

6.2.1. Choix des types de données

- Adaptez les types du MLD aux types spécifiques du SGBD.
  - Exemple : `INT` pour les identifiants, `VARCHAR(n)` pour les chaînes de caractères limitées, `TEXT` pour les textes longs.
- Prenez en compte la précision pour les types numériques (`DECIMAL`, `FLOAT`) et les dates (`DATE`, `DATETIME`, `TIMESTAMP`).

6.2.2. Contraintes d'intégrité

- **Clés primaires** pour identifier de manière unique chaque enregistrement.
- **Clés étrangères** pour maintenir l'intégrité référentielle entre les tables.
- **Contraintes uniques** pour garantir qu'une valeur (exemple : immatriculation d'une voiture) n'existe qu'une seule fois.
- **Contraintes de non-nullité** pour forcer la présence de données essentielles.

Exemple : Création d'une table en intégrant les contraintes d'intégrités

```
CREATE TABLE Livres (
    -- Clé primaire pour identifier chaque livre
    idLivre INT AUTO_INCREMENT PRIMARY KEY,
    -- Contraintes de non-nullité : il doit avoir un titre
    titre VARCHAR(255) NOT NULL,
    -- Contraintes de non-nullité : il doit avoir un auteur
    auteur VARCHAR(255) NOT NULL,
    -- Contrainte unique et de non-nullité :
    -- chaque ISBN est unique et obligatoire
    isbn VARCHAR(13) UNIQUE NOT NULL,
    -- Année de publication (peut être facultative)
    anneePublication YEAR,
     -- Clé étrangère pour l'intégrité référentielle avec la table
Categories
    idCategorie INT,
    -- Clé étrangère pour lier chaque livre à une catégorie
    -- Comportement : si la catégorie est supprimée, mettre
    -- idCategorie à NULL
    CONSTRAINT fk_categorie FOREIGN KEY (idCategorie)
        REFERENCES Categories(idCategorie)
        ON DELETE SET NULL
);
```

Explication des éléments utilisés

1. **Clés primaires** :
   - La colonne `idLivre` est définie comme la clé primaire. Elle garantit l'unicité de chaque enregistrement et permet une identification rapide.
   - L'attribut `AUTO_INCREMENT` permet d'attribuer automatiquement une valeur unique à chaque nouveau livre.
2. **Clés étrangères** :
   - La colonne `idCategorie` est une clé étrangère qui relie chaque livre à une catégorie dans la table **Categories**.
   - Cela garantit que la catégorie associée à un livre existe réellement. Si une catégorie est supprimée, la contrainte `ON DELETE SET NULL` met à jour les livres concernés pour ne plus avoir de catégorie.
3. **Contraintes uniques** :
   - La colonne `isbn` a une contrainte `UNIQUE`, car chaque livre publié doit avoir un ISBN unique. Cela évite les doublons dans la base de données. Il pourrait-être utilisé comme clé primaire.
4. **Contraintes de non-nullité** :
   - Les colonnes `titre`, `auteur`, et `isbn` ont la contrainte `NOT NULL`, car ces informations sont essentielles pour identifier un livre.
   - Cela empêche l'insertion de livres sans ces données obligatoires.

Exemple d'insertion et d'intégrité référentielle

- **Insertion correcte**

```
INSERT INTO Livres (titre, auteur, isbn, anneePublication, idCategorie)
VALUES ('Le Petit Prince', 'Antoine de Saint-Exupéry', '9782070612758', 1943, 1);
```

- **Tentative d'insertion incorrecte : Violation de contrainte**

```
-- Erreur : Violation de la contrainte UNIQUE sur isbn
INSERT INTO Livres (titre, auteur, isbn, anneePublication, idCategorie)
VALUES ('Livre en doublon', 'Auteur', '9782070612758', 2000, 2);
```

Si un livre est inséré sans ISBN ou avec un ISBN déjà existant, une erreur est déclenchée.

- **Impact d'une suppression dans la table *Categories***

Si la catégorie avec `idCategorie = 1` est supprimée, les livres associés auront leur `idCategorie` mis à `NULL` :

```
DELETE FROM Categories WHERE idCategorie = 1;
```

6.2.3. Indexation

L'indexation est un mécanisme essentiel dans la conception physique d'une base de données relationnelle. Elle permet d'améliorer considérablement les performances des requêtes en réduisant le temps nécessaire pour accéder aux données stockées dans les tables.

Fonctionnement de l'indexation

Un index est une structure de données auxiliaire, généralement basée sur un arbre (comme un B-Tree) ou une table de hachage, qui stocke des références à des enregistrements de la table en fonction des valeurs d'une ou plusieurs colonnes. Ces structures permettent de localiser rapidement les enregistrements sans avoir à parcourir toute la table.

Types d'index

- **Index primaire :**

  L'index primaire est automatiquement créé sur les colonnes définies comme clé primaire d'une table. Il garantit l'unicité des valeurs et permet un accès rapide aux enregistrements via la clé primaire.

- **Index unique :**

  Semblable à l'index primaire, l'index unique empêche les doublons dans une ou plusieurs colonnes, mais il n'est pas nécessairement utilisé comme clé principale.

- **Index secondaire :**

  Les index secondaires sont créés sur des colonnes fréquemment utilisées dans les clauses `WHERE`, `JOIN`, ou `ORDER BY`. Ils permettent d'accélérer l'exécution des requêtes.

- **Index composite :**

  Ces index sont créés sur plusieurs colonnes et sont particulièrement utiles pour optimiser les requêtes impliquant des combinaisons spécifiques de colonnes.

Rôle des index dans notre exemple de bibliothèque

- La table **Emprunts** contient deux clés étrangères : l'identifiant du lecteur (`idLecteur`) et l'identifiant du livre (`idLivre`). Des index peuvent être créés sur ces colonnes pour accélérer les requêtes telles que :

  ```
  SELECT * FROM Emprunts WHERE idLecteur = ? ;
  SELECT * FROM Emprunts WHERE idLivre = ? ;
  ```

  Cela permet de localiser rapidement les emprunts pour un lecteur donné ou de retrouver les emprunts associés à un livre spécifique.

- Un index composite pourrait être créé sur les colonnes `idLecteur` et `idLivre` pour optimiser les requêtes impliquant ces deux colonnes ensemble, comme :

  ```
  SELECT * FROM Emprunts WHERE idLecteur = ? AND idLivre = ? ;
  ```

Considérations sur l'indexation

- **Avantages** :
  - Accélération des recherches et des tris.
  - Réduction des temps d'accès pour les jointures.
- **Inconvénients** :
  - Les index augmentent l'espace de stockage requis.
  - Ils ralentissent les opérations d'insertion, de mise à jour et de suppression, car les index doivent être maintenus à jour.

Stratégie d'indexation

Lors de la conception du niveau physique, il est crucial d'analyser les cas d'utilisation principaux de la base de données pour déterminer les colonnes nécessitant une indexation. Cela implique :

- Identifier les colonnes les plus souvent utilisées dans les requêtes.
- Éviter les index inutiles ou redondants, qui augmenteraient les coûts de maintenance sans bénéfice notable.

6.2.4. Optimisation du stockage

- La Taille des champs doit être adaptée pour éviter le gaspillage de mémoire (exemple : `VARCHAR(50)` au lieu de `TEXT` si la taille maximale est connue).
- Des stratégies de compression peuvent être implémentées pour les colonnes de grandes tables et, si le SGBD le permet.

6.3. Gestion des transactions

La gestion des transactions représente des mécanismes pour garantir que les opérations complexes soient **atomiques**, **cohérentes**, **isolées**, et **durables** (propriétés ACID).

Contexte de l'exemple

Lorsqu'un lecteur emprunte un livre, deux opérations doivent être réalisées :

1. Ajouter une entrée dans la table **Emprunts** avec les informations sur l'emprunt.
2. Mettre à jour la table **Livres** pour indiquer que le livre est désormais indisponible.

Mise en œuvre des propriétés ACID

1. **Atomicité**
   Une transaction est atomique si toutes les opérations qui la composent réussissent ou échouent ensemble.

   Exemple :
   Si l'insertion dans la table **Emprunts** réussit, mais que la mise à jour de la table **Livres** échoue, la transaction entière est annulée pour éviter des incohérences.

```sql
BEGIN TRANSACTION;

INSERT INTO Emprunts (idLecteur, idLivre, dateEmprunt, dateRetourPrevu)
VALUES (1, 101, '2025-01-15', '2025-02-15');

UPDATE Livres
SET disponible = FALSE
WHERE idLivre = 101;

COMMIT; -- Si tout réussit, confirme les changements.
-- ROLLBACK; -- Si une opération échoue, annule tout
```

2. **Cohérence :** Une transaction garantit que la base passe d'un état valide à un autre, respectant toutes les contraintes (clés primaires, étrangères, etc.).

   Exemple :

   Si le livre n'existe pas dans la table **Livres** ou si le lecteur n'est pas enregistré dans la table **Lecteurs**, la transaction échoue. Cela garantit l'intégrité référentielle.

   Illustration SQL :

```sql
INSERT INTO Emprunts (idLecteur, idLivre, dateEmprunt, dateRetourPrevu)
VALUES (1, 999, '2025-01-15', '2025-02-15');
-- Erreur : la clé étrangère idLivre = 999 ne correspond à aucun livre existant
```

3. **Isolation :** Les transactions concurrentes ne doivent pas interférer les unes avec les autres.

   Exemple :

   Si deux utilisateurs essaient d'emprunter le même livre simultanément, un mécanisme de verrouillage (lock) empêche qu'ils puissent réserver le même exemplaire.

   Illustration SQL :

```sql
BEGIN TRANSACTION;
-- Verrouillage pour empêcher d'autres transactions d'accéder à ce livre
SELECT * FROM Livres WHERE idLivre = 101 FOR UPDATE;
INSERT INTO Emprunts (idLecteur, idLivre, dateEmprunt, dateRetourPrevu)
VALUES (1, 101, '2025-01-15', '2025-02-15');
UPDATE Livres SET disponible = FALSE WHERE idLivre = 101;
COMMIT;
```

4. **Durabilité :** Une fois qu'une transaction est validée (committed), ses effets doivent persister même en cas de panne système.

   Exemple :

   Après qu'un lecteur a emprunté un livre, si une coupure de courant survient, les modifications (ajout dans **Emprunts**, mise à jour de **Livres**) doivent être sauvegardées. Cela est généralement assuré par un journal de transactions (transaction log).

   Illustration :

   Les systèmes de bases de données comme MySQ enregistrent chaque transaction dans un journal avant de l'appliquer, garantissant ainsi qu'elle peut être rejouée en cas de problème.

6.4. Sécurité et droits d'accès

Implémentez des droits d'accès granulaires pour contrôler qui peut lire, écrire, ou modifier certaines données.

Illustration de la Sécurité et des Droits d'Accès

Les droits d'accès granulaires permettent de contrôler précisément les actions que chaque catégorie d'utilisateur peut effectuer sur la base de données. Cela garantit la confidentialité, l'intégrité et la sécurité des données en limitant l'accès aux seules informations nécessaires.

Contexte de l'exemple :

Prenons l'exemple du système de gestion de bibliothèque pour illustrer l'application de ces principes. Dans une bibliothèque, il existe différents types d'utilisateurs avec des besoins spécifiques :

1. **Administrateurs** : Gestion complète des données (ajout, modification, suppression).
2. **Bibliothécaires** : Lecture et modification des emprunts, mais pas des informations des lecteurs.
3. **Lecteurs** : Consultation de leurs propres emprunts, sans pouvoir les modifier ni accéder aux données des autres utilisateurs.

Mise en œuvre des droits d'accès

1. **Création de rôles :** Définissez des rôles dans le système de gestion de base de données pour chaque type d'utilisateur.

   Exemple SQL :

```
-- Création des rôles
CREATE ROLE admin;
CREATE ROLE bibliothecaire;
CREATE ROLE lecteur;
```

2. **Définition des droits pour chaque rôle :**Attribuez des permissions spécifiques à chaque rôle en fonction de ses besoins.

   Exemple SQL :

```
-- Permissions pour les administrateurs
GRANT ALL PRIVILEGES ON ALL TABLES IN SCHEMA public TO admin;

-- Permissions pour les bibliothécaires
GRANT SELECT, INSERT, UPDATE ON TABLE Emprunts TO bibliothecaire;
GRANT SELECT ON TABLE Lecteurs TO bibliothecaire; -- Lecture seule

-- Permissions pour les lecteurs
GRANT SELECT ON TABLE Emprunts TO lecteur;
-- Pas d'accès à la table Lecteurs
REVOKE SELECT ON TABLE Lecteurs FROM lecteur;
```

3. **Restrictions au niveau des données :**Pour les utilisateurs ayant accès à des parties limitées des données, ajoutez des filtres ou des vues.

   Exemple : Création d'une vue pour les emprunts d'un lecteur spécifique :

```
CREATE VIEW Emprunts_Utilisateur AS
SELECT * FROM Emprunts
WHERE idLecteur = CURRENT_USER;
GRANT SELECT ON Emprunts_Utilisateur TO lecteur;
```

4. **Audit des actions :**Configurez des journaux pour surveiller l'activité des utilisateurs, garantissant ainsi une traçabilité des actions effectuées.

   Exemple SQL :

```
-- Activer l'audit des modifications sur la table Emprunts
CREATE TRIGGER audit_trigger
AFTER UPDATE OR DELETE ON Emprunts
FOR EACH ROW
EXECUTE FUNCTION log_modifications();
```

Scénario complet : Lecture et modification des emprunts

- **Un bibliothécaire** consulte et modifie un emprunt :

  Il a les permissions nécessaires sur la table **Emprunts**, mais ne peut pas supprimer de données sensibles comme les informations des lecteurs.

  Requête possible :

```
UPDATE Emprunts SET dateRetourReel = '2025-01-20' WHERE idEmprunt = 5;
```

- **Un lecteur** consulte ses propres emprunts via la vue :

  Il ne peut accéder qu'aux données le concernant, grâce à la vue définie.

  Requête possible :

```
DELETE FROM Lecteurs WHERE idLecteur = 10;
```

- **Un administrateur** effectue une suppression ou une opération critique :

  Il dispose de privilèges complets pour gérer toutes les tables.

  Requête possible :

```
DELETE FROM Lecteurs WHERE idLecteur = 10;
```

Résumé

- ***Contrôle d'accès granulaire :*** *Les permissions sont adaptées aux besoins spécifiques de chaque rôle.*
- **Sécurité renforcée** : Les utilisateurs ne peuvent pas accéder ou modifier des données non autorisées.
- **Traçabilité** : Les actions sont auditées pour détecter tout accès ou modification non autorisés.
- **Adaptabilité** : L'utilisation de vues et de rôles permet une flexibilité dans la gestion des droits.