64-42.1 Projet de manipulation de documents

Rapport pour le projet individuel

Travail individuel

Nom et prénom : Ameli Darwin
Date de restitution : 07.04.2024

Table des matières

1 Introduction

Le présent rapport décrit en détail mon projet individuel de gestion d'accès pour des lieux par des personnes, réalisé dans le cadre du cours "Projet de manipulation de documents". Ce projet vise à mettre en œuvre une solution complète de gestion d'accès, où les utilisateurs peuvent accéder à différents lieux à l'aide de clés, tout en assurant une traçabilité des accès et des modifications.

Initialement, j'ai cherché à concevoir un système permettant la gestion des accès à des lieux par des personnes, avec la possibilité de définir des règles de verrouillage et de déverrouillage, ainsi que de suivre les historiques des accès et des modifications. Pour répondre à ce besoin, j'ai effectué une analyse approfondie des entités impliquées, me permettant de définir un modèle de données relationnel adapté à la gestion de ces informations.

Mon projet est structuré autour de plusieurs entités principales :

- Clef : Représente une clé d'accès, caractérisée par un numéro de série et un statut.
- Personne : Définit les utilisateurs du système, avec leur nom, prénom, email et groupe d'appartenance.
- Groupe : Regroupe des personnes et permet de définir des accès collectifs à des lieux.
- Accès : Gère les autorisations d'accès des groupes à des lieux, avec des dates de début et de fin.
- Lieu : Désigne les endroits physiques où les accès sont autorisés, pouvant être composés de sous-lieux.
- Historique : Trace les accès et les modifications effectuées dans le système, fournissant une traçabilité complète des actions réalisées.

Afin de répondre aux besoins spécifiques du projet, j'ai défini des contraintes pour garantir la cohérence et l'intégrité des données. Par exemple, j'ai mis en place des contraintes sur les statuts des clés et des contraintes de dates pour les accès, afin d'assurer un contrôle efficace des autorisations d'accès.

Ce rapport présentera en détail les différentes étapes de réalisation du projet, notamment l'analyse des besoins, la modélisation des données, l'implémentation du modèle relationnel en base de données, la génération et le stockage du document XML, ainsi que l'extraction du document XML vers un fichier au format XML. Chaque étape sera expliquée en détail, avec des exemples de code, des descriptions des processus et des choix techniques.

2 Contraintes

La conception et la mise en œuvre du projet de gestion d'accès pour des lieux par des personnes ont été réalisées en tenant compte de plusieurs contraintes visant à assurer la qualité, la cohérence et l'intégrité des données manipulées. Ces contraintes ont été définies en fonction des besoins spécifiques du projet et des bonnes pratiques de gestion de données. Voici les principales contraintes prises en compte :

Contraintes sur les statuts des clés :

- Les clés peuvent avoir l'un des statuts suivants : ACTIVE, INACTIVE, PERDUE ou DYSFONCTIONNELLE.
- Chaque clé doit obligatoirement avoir un statut valide parmi les options définies.

Contraintes de cohérence des données :

- Chaque personne doit être associée à un groupe existant dans le système.
- Chaque accès doit être lié à un groupe et à un lieu existant dans la base de données.
- Les dates de début et de fin d'un accès doivent être cohérentes et dans un ordre chronologique valide.
- Les lieux peuvent être composés de sous-lieux, mais une structure hiérarchique correcte doit être maintenue pour garantir l'intégrité des données.
-

Contraintes de validité des données :

- Le nom et le prénom d'une personne ne peuvent pas être vides.
- L'adresse email d'une personne doit être unique et respecter un format valide.
- Les numéros de série des clés doivent être uniques pour chaque clé.
- Les noms des groupes et des lieux ne peuvent pas être vides et doivent être uniques dans le système.

Contraintes de gestion des accès :

- Chaque personne peut posséder une seule clé à la fois.
- Un groupe peut avoir des accès à plusieurs lieux, mais une gestion fine des autorisations est mise en place pour chaque accès.

Ces contraintes ont été intégrées dans la conception du modèle de données relationnel ainsi que dans les scripts SQL de création de tables et de contraintes. Elles permettent de garantir la fiabilité et la robustesse du système de gestion d'accès tout au long de son utilisation.

3 Analyse

Dans cette section, nous allons examiner en détail le modèle de données conçu pour notre projet de gestion d'accès pour des lieux par personnes. Cette analyse vise à expliquer les choix conceptuels et structurels effectués lors de la conception du modèle de données, ainsi que leur adéquation avec les besoins spécifiques du projet.

3.1 Modèle Conceptuel de Données (MCD)

Le Modèle Conceptuel de Données (MCD) constitue la première étape de la conception d'une base de données. Il permet de décrire les entités, les relations et les contraintes qui vont structurer les données du système. Notre MCD a été conçu en utilisant la méthode d'analyse Merise, et il est représenté graphiquement sous forme de diagramme entité-relation (ER).

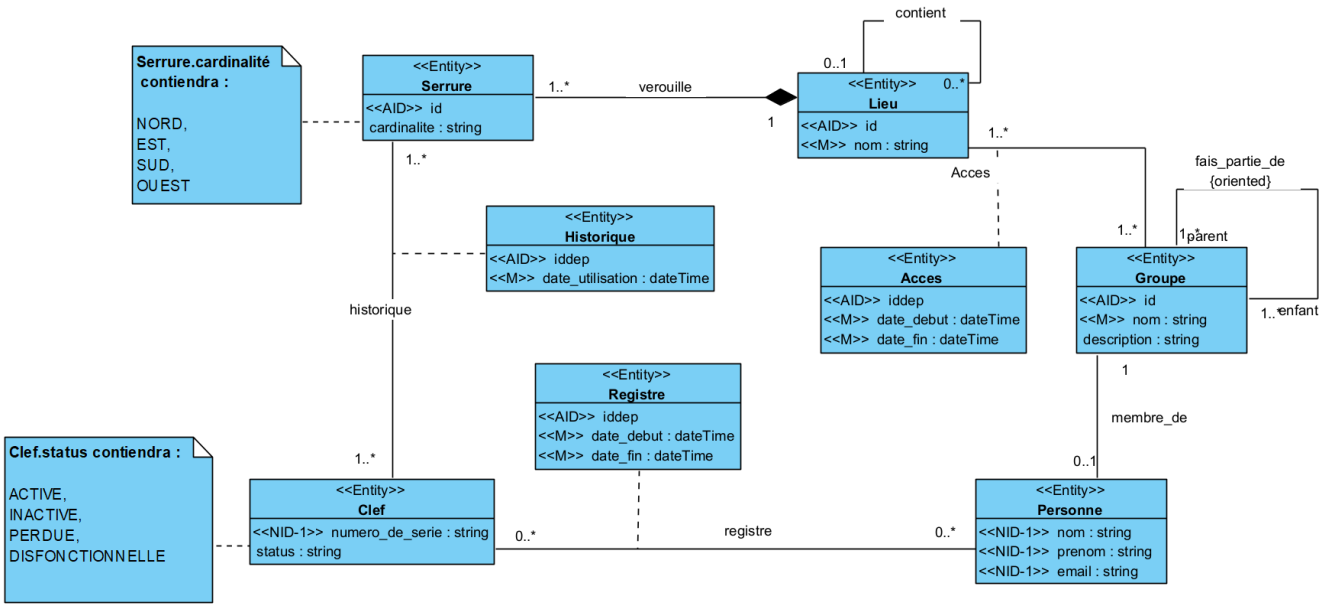


Figure 1 MCD pour référence

Entités principales :

- Personne : Représente les utilisateurs du système, avec des attributs tels que nom, prénom et email.
- Clé : Représente les clés d'accès aux différents lieux, avec des attributs comme le numéro de série et le statut.
- Groupe : Représente les groupes d'utilisateurs, avec des attributs comme le nom et la description.
- Lieu : Représente les lieux auxquels les accès sont autorisés, avec un attribut pour le nom.
- Accès : Représente les autorisations d'accès accordées à chaque groupe pour chaque lieu, avec des attributs pour les dates de début et de fin de l'accès.

Relations principales :

- Possède : Relation entre Personne et Clé, indiquant quelle personne possède quelle clé.
- Membre : Relation entre Personne et Groupe, indiquant à quel groupe appartient chaque personne.
- Participe : Relation entre Groupe et Accès, indiquant quels groupes ont accès à quels lieux.
- Composé_de : Relation récursive entre Lieu et Lieu, permettant de modéliser une hiérarchie de lieux composés de sous-lieux.

3.2 Choix des données et des types

Les données ont été sélectionnées en fonction des besoins fonctionnels du système de gestion d'accès. Par exemple, le choix de stocker les statuts des clés et les dates d'accès permet de suivre l'état des autorisations et d'enregistrer l'historique des accès. De plus, des contraintes de validation ont été définies pour garantir la cohérence et l'intégrité des données, telles que l'unicité des emails des personnes et des numéros de série des clés.

3.3 Justification des choix

Les choix opérés lors de la conception du modèle de données ont été influencés par les principes fondamentaux de la normalisation des bases de données, notamment les formes normales, ainsi que par les meilleures pratiques de modélisation. L'approche adoptée visait à réduire la redondance des données et à maintenir la cohérence du système, en mettant en place des relations entre les entités. Par exemple, en optant pour la gestion des autorisations par groupe, inspirée du modèle Active Directory de Microsoft, j'ai pu bénéficier d'une flexibilité et d'une évolutivité accrues en matière de sécurité des accès.

Initialement, j'avais attribué les accès directement aux personnes, ce qui a rendu la gestion des ajouts et des restrictions d'accès plus complexe. Cependant, en m'inspirant du système de droits d'accès de Linux, j'ai adopté une approche expérimentale qui s'est avérée très efficace. Cette approche repose sur l'entité Groupe, conçue de manière récursive pour éviter les limitations dans la complexité hiérarchique de la structure requise.

Afin de simplifier la gestion des entités, j'ai opté pour une stratégie similaire à celle de Linux, où le groupe enfant possède un nom identique à celui de la personne. Cette approche permet d'ajouter les droits d'accès à une personne soit en passant par ses groupes parent, soit en lui attribuant directement des autorisations au sein de son propre groupe (avec une description unique pour éviter les duplications dans le nom du groupe).

4 Implémentation du modèle de données en base de données relationnelle

Dans cette section, nous détaillerons l'implémentation du modèle de données dans une base de données relationnelle. Nous aborderons les différentes étapes de cette implémentation, de la conception initiale à la création des tables et des contraintes dans Oracle Database.

4.1 Conception du Modèle de Données

Nous avons utilisé l'outil de modélisation Visual Paradigm pour concevoir notre modèle de données. Cet outil nous a permis de créer le Modèle Conceptuel de Données (MCD), le Modèle Logique de Données (MLD) et le Modèle Physique de Données (MPD). Chaque modèle correspond à une étape spécifique du processus de conception et sert de référence pour la création ultérieure de la base de données.

4.2 Modèle Conceptuel de Données (MCD)

Le MCD, représenté graphiquement sous forme de diagramme entité-relation (ER), fournit une vue conceptuelle des entités, des relations et des contraintes du système. Il s'agit d'une abstraction haut niveau qui permet de comprendre la structure générale des données sans se soucier des détails d'implémentation.

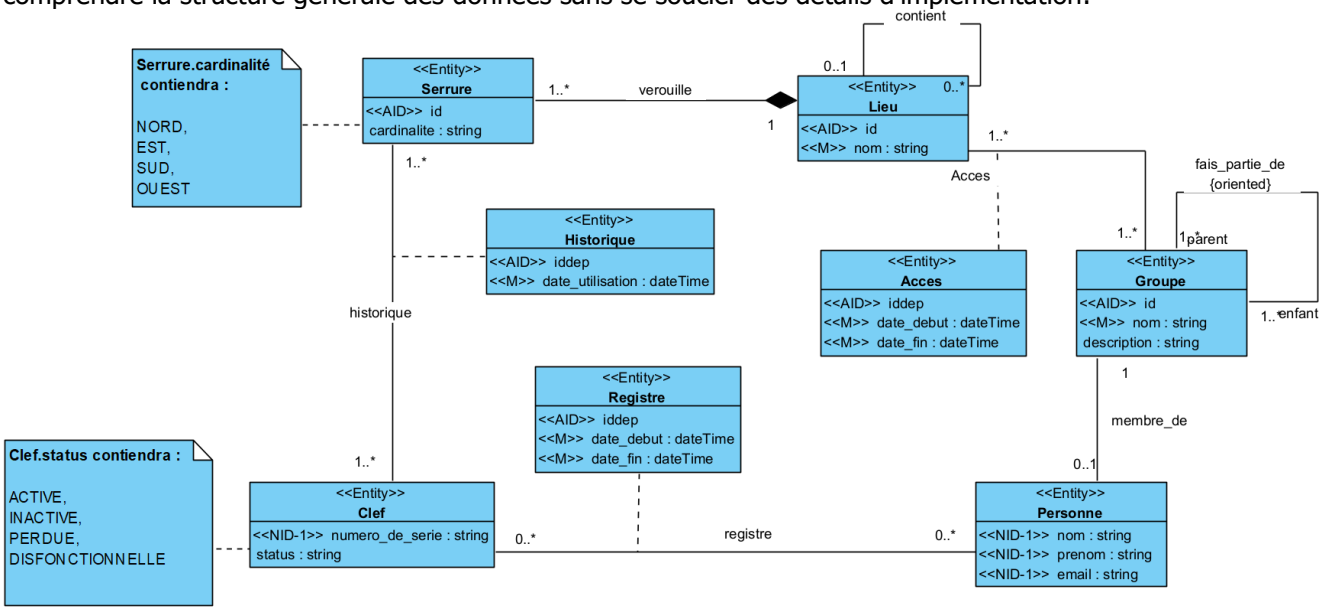


Figure 2 MCD

4.3 Modèle Logique de Données (MLD)

Le MLD traduit le MCD en termes plus précis, en définissant les entités, les attributs et les relations sous forme de schéma relationnel. Il spécifie également les clés primaires, les clés étrangères et les contraintes d'intégrité référentielle nécessaires à la mise en œuvre du modèle dans une base de données relationnelle.

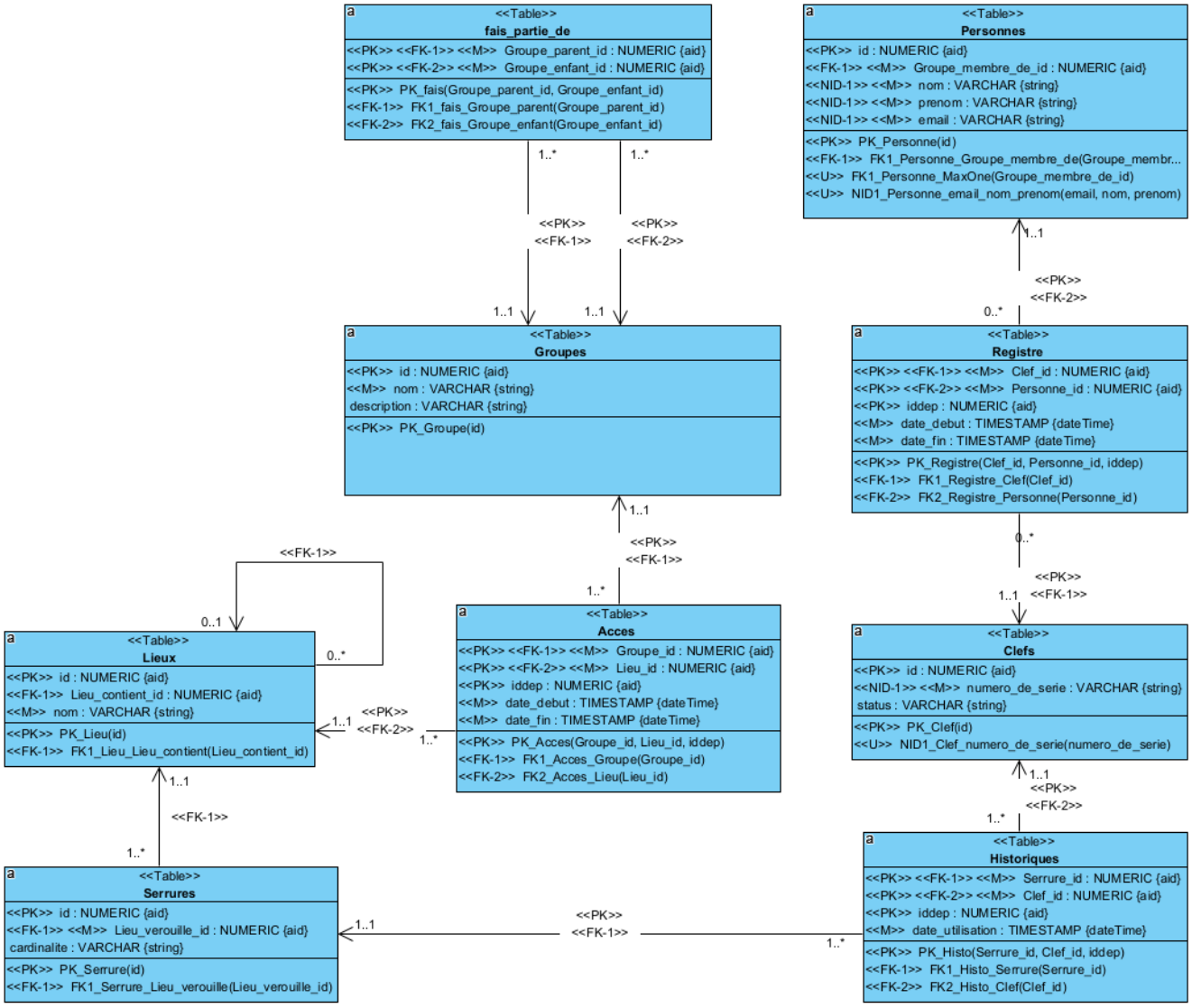


Figure 3 MLD

4.4 Modèle Physique de Données (MPD)

Le MPD représente la structure physique réelle de la base de données, y compris les types de données spécifiques, les index, les partitions et autres propriétés de stockage. Il s'agit du modèle final qui sera implémenté dans le système de gestion de base de données. Ici le MPD correspond à ce que je vais implémenter dans Oracle.

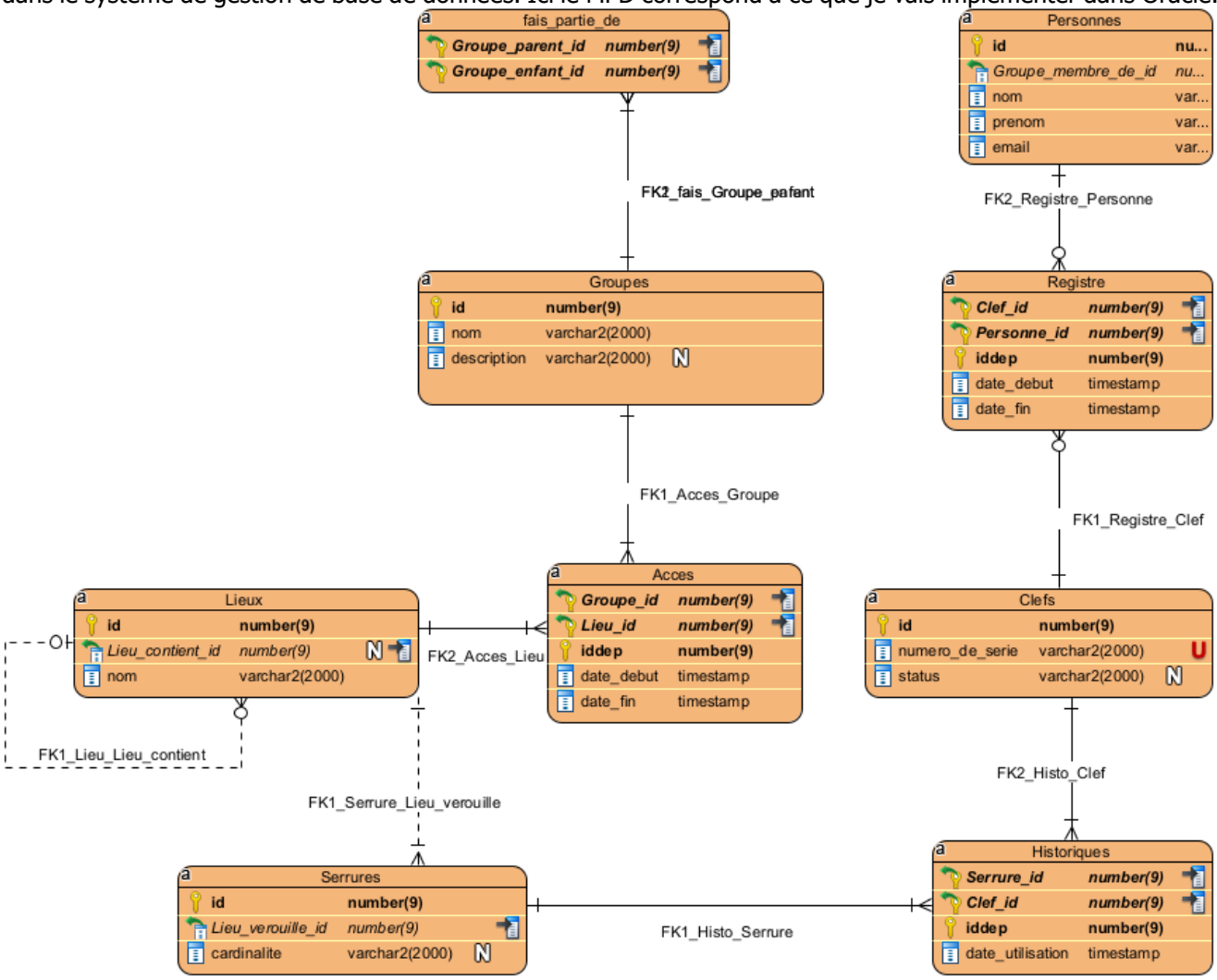


Figure 4 MPD

4.5 Adaptation des Scripts SQL

Les scripts SQL générés par Visual Paradigm ont été adaptés pour répondre aux spécificités d'Oracle Database. Cela inclut la définition des types de données Oracle, l'ajout de contraintes CHECK pour garantir l'intégrité des données, et d'autres ajustements nécessaires pour assurer la compatibilité avec le système de gestion de base de données cible. Merci Visual Paradigm d'enlever cette épine du pied.

Les scripts de création de la base sont disponibles en annexe.

5 Document XML

Dans cette section, nous aborderons le document XML généré dans le cadre de notre projet de gestion d'accès pour des lieux par des personnes.

5.1 Objectif du Document XML

Le document XML constitue une représentation structurée des données du système de gestion d'accès. Son objectif principal est de fournir une vue portable et standardisée des informations relatives aux groupes, aux personnes, et à leurs attributs associés. Ce document XML capture les données hiérarchiques des groupes et des personnes, permettant ainsi une représentation claire de la structure de l'organisation et des membres associés à chaque groupe.

5.2 Contenu du Document XML

```
<Groupes>
	<Groupe ID="1" NomGroupe="Gestion"></Groupe>
	<Groupe ID="1" NomGroupe="Ingénierie"></Groupe>
	<Groupe ID="2" NomGroupe="Petit">
		<Personnes>
			<Personne>
				<ID>2</ID>
				<Nom>Petit</Nom>
				<Prenom>Jean</Prenom>
				<Email>jean.petit@he-arc.ch</Email>
			</Personne>
		</Personnes>
	</Groupe>
	<Groupe ID="3" NomGroupe="Durand">
		<Personnes>
			<Personne>
				<ID>1</ID>
				<Nom>Durand</Nom>
				<Prenom>Sophie</Prenom>
				<Email>sophie.durand@he-arc.ch</Email>
			</Personne>
		</Personnes>
	</Groupe>
</Groupes>
```

Le document XML contient les informations suivantes :

- Groupes : Chaque groupe est représenté par un élément <Groupe> contenant son identifiant (ID) et son nom de groupe (NomGroupe).
- Personnes : Les personnes appartenant à chaque groupe sont listées dans un élément <Personnes> sous le groupe correspondant. Chaque personne est représentée par un élément <Personne> contenant son identifiant (ID), son nom, son prénom et son adresse email. Vu les groupes persos ont aurait pu se passer de <Personnes>, mais j'ai pensé aux enseignant qui pouvait avec plusieurs « personnes dans un groupe perso ».

Le XML ne correspond pas exactement à ce que je souhaitais faire. En effet la recursivité de Groupe n'est pas respecté ici.

Le XML cible aurait été :

```
<Groupes>
	<Groupe ID="1" NomGroupe="HE-Arc">
		<Groupe ID="2" NomGroupe="Gestion">
```

```
                <Groupe ID="5" NomGroupe="Petit">
                      <Personnes>
                            <Personne>
                                  <ID>2</ID>
                                  <Nom>Petit</Nom>
                                  <Prenom>Jean</Prenom>
                                  <Email>jean.petit@he-arc.ch</Email>
                            </Personne>
                      </Personnes>
                </Groupe>
          </Groupe>
          <Groupe ID="3" NomGroupe="Ingénierie">
                <Groupe ID="4" NomGroupe="Durand">
                      <Personnes>
                            <Personne>
                                  <ID>1</ID>
                                  <Nom>Durand</Nom>
                                  <Prenom>Sophie</Prenom>
                                  <Email>sophie.durand@he-arc.ch</Email>
                            </Personne>
                      </Personnes>
                </Groupe>
          </Groupe>
      </Groupe>
</Groupes>
```

On voit la différence de hiérarchie dans les groupes. Dans le second exemple elle est respectée. Malheureusement je n'ai pas réussi à faire de sous-requêtes dans la clause XMLForest ce qui m'empêche de terminer ma structure récursive de Groupe.

5.3 Structure du Document XML

Le document XML suit une structure hiérarchique, où chaque groupe est représenté par un élément XML distinct. Les éléments <Personnes> sont inclus à l'intérieur des éléments <Groupe>, reflétant ainsi la relation hiérarchique entre les groupes et les personnes. Cette structure permet une visualisation claire des membres associés à chaque groupe et de leur appartenance respective.

6 Schéma XML

Dans cette section, nous examinerons le schéma XML associé à notre projet de gestion d'accès pour des lieux par des personnes. Le schéma XML, au format XSD (XML Schema Definition), définit la structure et les contraintes des données contenues dans le document XML. Nous décrirons comment le schéma a été conçu, quelles contraintes il impose et comment il garantit la cohérence des données.

6.1 Conception du Schéma XML

Le schéma XML a été conçu pour refléter étroitement la structure des données du système de gestion d'accès. Il définit deux types de données principaux : "GroupeType" et "PersonneType". Ces types permettent de représenter les groupes et les personnes, respectivement.

6.2 Structure du Schéma XML

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="Groupes">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="Groupe" type="GroupeType" maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

  <xs:complexType name="GroupeType">
    <xs:sequence>
      <xs:element name="Personnes" minOccurs="0">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="Personne" maxOccurs="unbounded">
              <xs:complexType>
                <xs:sequence>
                  <xs:element name="ID" type="xs:integer"/>
                  <xs:element name="Nom" type="xs:string"/>
                  <xs:element name="Prenom" type="xs:string"/>
                  <xs:element name="Email" type="xs:string"/>
                </xs:sequence>
              </xs:complexType>
            </xs:element>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
      <xs:element name="Groupe" type="GroupeType" minOccurs="0"
maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attribute name="ID" type="xs:integer" use="required"/>
    <xs:attribute name="NomGroupe" type="xs:string" use="required"/>
  </xs:complexType>
</xs:schema>
```

Le schéma XML se compose principalement de deux types de données complexes :

- Type de données "GroupeType" : Ce type représente un groupe dans le système. Il contient une séquence d'éléments, comprenant un identifiant (ID), un nom de groupe (NomGroupe), et éventuellement une liste de personnes appartenant à ce groupe, ainsi que des sous-groupes.

- Type de données "PersonneType" : Ce type représente une personne dans le système. Il contient un identifiant (ID), un nom, un prénom et une adresse email.

Chaque groupe peut contenir une liste de personnes (éléments "Personnes") et une liste de sous-groupes (éléments "Groupe"). Cette structure permet une représentation hiérarchique des groupes et des personnes, avec la possibilité de sous-groupes imbriqués.

6.3 Contraintes du Schéma XML

Le schéma XML impose des contraintes sur les données pour garantir leur intégrité et leur cohérence. Par exemple, il spécifie que l'identifiant et le nom d'un groupe sont requis, et que l'identifiant d'une personne doit être un entier. De plus, il définit les types de données autorisés pour chaque attribut, par exemple, l'adresse e-mail doit être une chaîne de caractères.

6.4 Cohérence avec le Modèle Physique de Données

Le schéma XML est cohérent avec le modèle physique de données (MPD) de notre système de gestion d'accès. Les types de données du schéma XML correspondent aux entités du MPD, ce qui assure une représentation précise et complète des données dans le document XML.

7 Génération du document XML et son stockage

Dans cette section, je vais aborder la manière dont j'ai généré le document XML à partir des données de la base de données relationnelle, ainsi que son stockage pour une utilisation ultérieure. Je vais décrire les choix et les démarches que j'ai effectués pour cette génération et ce stockage, ainsi que les considérations techniques prises en compte.

7.1 Génération du document XML

Pour générer le document XML à partir des données de ma base de données, j'ai opté pour l'utilisation de requêtes SQL. J'ai envisagé plusieurs approches et j'ai choisi la meilleure solution en fonction de la complexité des données et des performances.

J'ai décidé d'utiliser des vues SQL pour structurer les données de manière appropriée avant de les exporter au format XML. Cela m'a permis de manipuler les données de manière flexible et de les préparer selon les besoins spécifiques du document XML. J'ai conçu des requêtes SQL pour récupérer les données de manière efficace et les organiser en fonction de la structure attendue du document XML. Ce que je défini dans le schema XSD.

```
-- Vue pour la Liste des Personnes par Groupe
CREATE OR REPLACE VIEW vue_personnes_groupes_xml AS
WITH hierarchy AS (
  SELECT  CONNECT_BY_ROOT f.Groupe_parent_id AS root_id,
          f.Groupe_enfant_id AS current_id,
          g.nom AS nom_groupe,
          LEVEL AS lvl
  FROM    fais_partie_de f
  JOIN    Groupes g ON f.Groupe_enfant_id = g.id
  CONNECT BY PRIOR f.Groupe_enfant_id = f.Groupe_parent_id
)
SELECT XMLElement("Groupes",
  XMLAgg(
    XMLElement("Groupe",
      XMLAttributes(h.root_id AS "ID", h.nom_groupe AS "NomGroupe"),
      XMLForest(
        (SELECT XMLAgg(
            XMLElement("Personne",
              XMLForest(p.id AS "ID",
                        p.nom AS "Nom",
                        p.prenom AS "Prenom",
                        p.email AS "Email")
            )
          )
          FROM Personnes p
          WHERE p.Groupe_membre_de_id = h.current_id
        ) AS "Personnes"
      )
    )
  )
).getClobVal() AS xml_data
FROM hierarchy h
WHERE h.lvl = 1;
```

Cette vue génère un document XML hiérarchique à partir des données des tables Groupes, fais_partie_de, et Personnes, elle utilise une CTE pour gérer la hiérarchie dans les données :

- CTE (Common Table Expression) - `hierarchy` :
  - La CTE `hierarchy` utilise la clause CONNECT_BY pour construire la hiérarchie des groupes à partir de la table fais_partie_de.
  - Elle sélectionne l'ID du groupe racine (root_id), l'ID du groupe actuel (current_id), le nom du groupe (nom_groupe), et le niveau dans la hiérarchie (lvl).

- Requête principale :

- Elle génère un document XML avec un élément racine Groupes.
- Pour chaque groupe, elle crée un élément Groupe avec les attributs ID et NomGroupe.
- Elle inclut également les personnes associées à chaque groupe en utilisant l'élément Personnes.

Ce document XML est structuré de manière à refléter la hiérarchie des groupes et à inclure les informations sur les personnes associées à chaque groupe.

	NUMERO	XML_DATA	GENERATION_TIMESTAMP	DESCRIPTION
1	9	<Groupes><Gro...	07.04.24 20:59:42.272554000	Liste des personnes par groupes
2	10	<Groupes><Gro...	07.04.24 22:45:44.917307000	Liste des personnes par groupes

Figure 5 Affichage depuis sqldev du contenu de la table journal_xml

7.2 Stockage du document XML

Après avoir généré le document XML, j'ai pris des décisions importantes quant à sa préservation et sa disponibilité à long terme. Pour garantir une gestion efficace, j'ai choisi de stocker le document XML dans une colonne de type XMLType d'une table dédiée au sein de la base de données relationnelle.

Cette approche a été préférée afin d'assurer une organisation structurée du document XML au sein de la base de données, simplifiant ainsi son accès et sa manipulation ultérieure. De plus, en utilisant une colonne XMLType, j'ai pu exploiter les fonctionnalités avancées de manipulation des données XML offertes par la base de données Oracle, notamment la validation XML et les requêtes XPath.

Une description succincte du document XML s'est également avérée nécessaire, étant donné la prévision d'intégrer plusieurs vues à l'avenir. Ainsi, même en l'absence de données spécifiques, cette description permettrait une identification claire du contenu du document XML, facilitant ainsi sa gestion et son utilisation. Bien que pour le moment le document XML ne contienne que des listes de personnes par groupes, j'ai anticipé l'ajout d'autres données telles que l'historique d'utilisation des clés. La mise en place de descriptions distinctes pour chaque vue permettrait une distinction claire des données provenant de différentes sources, garantissant ainsi une meilleure organisation et une meilleure compréhension du document XML.

7.3 Gestion de la version du document XML

Pour assurer le suivi des versions du document XML, j'ai ajouté une colonne supplémentaire dans la table de stockage du document XML. Cette colonne enregistre la date de création. Ainsi, chaque version du document est associée à une date temporelle, ce qui facilite la traçabilité et la gestion des différentes versions du document.

Vu que je n'utilise pas d'outil de versionning c'est à chaque fois une nouvelle entrée qui est généré avec un CURRENT_TIMESTAMP appliquer. Pour optimiser le stockage dans la base de données il aurait fallu faire des différentielles de modifications sur un même champ pour économiser le nombre d'entrées.

7.4 Fréquence de génération du document XML

La fréquence de génération du document XML dépend des besoins spécifiques de l'application. Dans mon cas, j'ai décidé de générer le document XML de manière spontanée, à chaque mise à jour des données dans la base de données. Cette approche garantit que le document XML est toujours à jour et reflète avec précision l'état actuel des données. Si on a besoin d'économiser les transactions car les requêtes sont trop longues avec chaque sérialisation on peut automatiser un trigger qui appellerait la vue à intervalle périodique (C'est l'avantage d'avoir la procédure dans une vue).

7.5 Gestion des erreurs et des exceptions

À ce stade du projet, aucun mécanisme spécifique de gestion des erreurs et des exceptions n'a été implémenté. En raison des contraintes de temps et de priorités, l'accent a été mis principalement sur la prouesse technique de la sérialisation de la récursivité dans la génération du document XML. La mise en place de mécanismes de gestion des erreurs et des exceptions reste une considération importante pour les étapes futures du projet, et des évaluations sont prévues pour déterminer les meilleures pratiques à mettre en œuvre.

Pour pallier l'absence de mécanismes de gestion des erreurs et des exceptions, plusieurs approches auraient pu être envisagées. Tout d'abord, l'implémentation de mécanismes de validation des données en amont aurait permis de détecter les erreurs potentielles lors de la génération du document XML. Par exemple, des règles de validation pourraient être définies pour garantir la conformité des données aux spécifications du schéma XML. De plus, la mise en place de routines de gestion des erreurs aurait pu permettre de capturer et de traiter les exceptions de manière appropriée lors de la génération ou du stockage du document XML. Enfin, l'utilisation de journaux d'audit pour enregistrer les événements et les erreurs aurait facilité la traçabilité et l'analyse des problèmes survenus. Ces différentes stratégies auraient contribué à améliorer la robustesse et la fiabilité du système dans son ensemble.

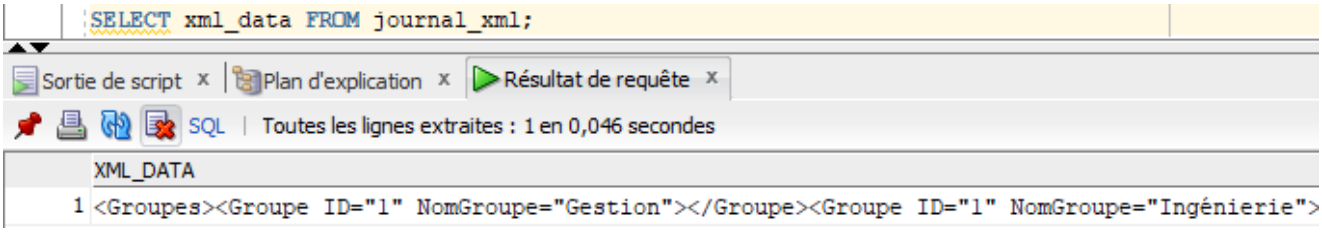
8 Extraction du document XML stocké en base de données vers un fichier au format XML

Une fois que le document XML a été généré et stocké en base de données, il est souvent nécessaire de l'extraire vers un fichier au format XML pour une utilisation ultérieure ou pour le partager avec d'autres systèmes. Dans cette section, nous aborderons le processus d'extraction du document XML stocké en base de données vers un fichier XML autonome.

8.1 Méthode d'extraction

Pour extraire le document XML stocké en base de données, nous utilisons une requête SQL qui récupère les données XML à partir de la base de données et les enregistre dans un fichier au format XML. Cette requête SQL est exécutée à l'aide d'un outil de gestion de base de données ou d'un langage de programmation tel que Python ou Java.

Voici un exemple de requête SQL utilisée pour extraire le document XML stocké en base de données vers un fichier XML : `SELECT xml_data FROM journal_xml;`



Dans cette requête, vue_personnes_groupes_xml est la vue qui sérialise le xml dans la table journal_xml. Cette table contient le document XML stocké en base de données, et xml_data est la colonne contenant les données XML. La fonction XMLAGG est utilisée pour agréger les données XML, et la fonction XMLSERIALIZE est utilisée pour sérialiser les données XML en format CLOB.

8.2 Ajout de la balise racine

Avant d'écrire les données XML dans un fichier, il est souvent nécessaire d'ajouter une balise racine pour encapsuler l'ensemble du document XML Cette balise racine, généralement définie par <?xml version="1.0"?>, encapsule l'ensemble du document XML. Cette étape peut être réalisée manuellement ou automatiquement à l'aide de la fonction XMLRoot lors de l'extraction des données XML.

Dans mes vues, j'ai fait le choix de ne pas inclure cette balise racine afin de maintenir la simplicité du xml data au sein des vues. Au lieu de cela, j'ai ajouté la balise root lors de l'importation des différents xml_data. Cette approche offre une flexibilité supplémentaire, notamment dans le cas où l'on souhaite fusionner les données de deux vues distinctes dans un seul fichier XML. Dans ce scénario, il serait plus simple d'ajouter la balise root après coup plutôt que de la retirer ou de la fusionner lors de l'importation initiale.

9 Conclusion

Ce projet de gestion d'accès a représenté une étape cruciale dans mon parcours d'apprentissage en matière de gestion de données et de développement logiciel. J'ai pu mettre en pratique les connaissances acquises en conception de bases de données relationnelles spécifique à Oracle, en modélisation XML, ainsi qu'en développement SQL avec ces intégrations XML auparavant j'étais toujours passé par un programme Java intermédiaire qui s'occupait du DAO.

9.1 Récapitulatif des réalisations

Durant ce projet, j'ai réussi à concevoir et implémenter un système de gestion d'accès robuste et fonctionnel. J'ai élaboré un modèle de données complet, comprenant des entités telles que les personnes, les clés, les groupes, les lieux, les accès et l'historique des utilisations de clés. Ce modèle a été traduit en un ensemble de tables relationnelles dans une base de données Oracle, avec toutes les contraintes nécessaires pour garantir l'intégrité des données.

De plus, j'ai généré un document XML structuré qui représente les données du système de gestion d'accès. Ce document XML peut être échangé avec d'autres systèmes ou applications nécessitant une intégration avec notre système. J'ai également conçu un schéma XML détaillé qui définit la structure et les contraintes des données contenues dans le document XML.

9.2 Perspectives d'amélioration

Bien que le projet actuel représente une première étape réussie, je suis conscient qu'il reste des possibilités d'amélioration et d'extension. J'aurais aimé explorer davantage les fonctionnalités de gestion de vues en base de données pour tester différents scénarios et cas d'utilisation. De même, j'aurais voulu élaborer des schémas XML plus complexes pour prendre en charge des fonctionnalités avancées telles que la validation des données et la transformation XSLT.

Le projet est entièrement disponible sur GitHub à l'adresse suivante : https://github.com/darwinamelihearc/MOD-clef. Vous y trouverez l'ensemble des commits, des requêtes SQL et des vues préparées pour le projet, même celles qui n'ont pas été implémentées dans la version finale. N'hésitez pas à explorer le code et à soumettre des suggestions ou des améliorations.

9.3 Conclusion finale

En conclusion, ce projet m'a permis de mettre en pratique mes compétences en gestion de données et en développement logiciel dans un contexte réel. J'ai appris énormément tout au long du processus, tant sur le plan technique que sur le plan de la gestion des priorités dans un projet. Je suis confiant que les connaissances et les expériences acquises me seront précieuses dans mes projets futurs.

Je tiens à remercier tous ceux qui m'ont soutenu et conseillé tout au long de ce projet, ainsi que les enseignants et intervenants qui y ont contribué.

10 Références

Modèle conceptuel de données – 2016 Pierre-André Sunier
Modèle conceptuel de données – 2018 Pierre-André Sunier
https://sgbd.developpez.com/tutoriels/cours-complet-bdd-sql/?page=normalisation#LVIII-E
https://docs.oracle.com/cd/B13789_01/server.101/b10759/functions203.htm
https://www.akadia.com/services/ora_gen_xml.html
https://docs.oracle.com/en/database/oracle/oracle-database/19/sqlrf/XMLFOREST.html
https://forums.oracle.com/ords/apexds/post/retrieve-data-from-recursive-xml-structure-8033
https://odieweblog.wordpress.com/2011/11/24/how-to-generate-a-recursive-xml-structure

11 Liste des documents transmis

- PMD-06-GENXML-Ameli_Darwin.sql
- PMD-06-MCD-Ameli_Darwin.pdf
- PMD-06-TABLE-Ameli_Darwin.sql
- PMD-06-XML-Ameli_Darwin.xml
- PMD-06-XSD-Ameli_Darwin.xsd
- Lien GitHub du projet : https://github.com/darwinamelihearc/MOD-clef
- ChatGPT pour la reformulation des paragraphes
- Copilot pour de la correction d'erreurs