

Rapport de projet :

NoSQL

Implémentation d'un modèle de données d'une application similaire à
l'application Elia

HAOUILI Ahmed – MOUHOUBI Abdeoulheb

SQL, noSQL et newSQL

Encadré par : M^{me} Maude Manouvrier

M2 IF App 2020 - 2021

Table des matières

1. Introduction	3
2. Modélisation du multi-modèle	3
2.1 Les données relationnelles	4
2.1.1 Données personnelles.....	4
2.1.2 Logins & mots de passe.....	5
2.1.3 Documents sur la plateforme	6
2.1.4 Liens utiles	8
2.2 Les données orientées documents.....	8
2.2.1 Informations de connexion.....	8
2.2.2 Messagerie.....	9
2.2.3 Absences des étudiants	10
2.2.4 Formations.....	10
2.2.5 Questionnaires.....	11
2.3 Les données orientées colonnes	11
2.3.1 Notes des étudiants	11
3. Implémentation du multi-modèle	12
3.1 Les données relationnelles	12
3.2 Les données orientées document	13
3.3 Les données orientées colonne.....	14
4. Conclusion.....	15

1. Introduction

Dans le cadre de l'UE : SQL, noSQL et newSQL, dispensée pour les étudiants de M2 Informatique pour la finance en apprentissage. Nous avons travaillé en binôme sur un projet, il s'agit de mettre en place un multi-modèle de données. Ce modèle devra mélanger des données relationnelles, ainsi que des données clé-valeur, colonne, document et graphe. Et cela dans le but de gérer l'accès aux données d'une application similaire à l'application ELIA. Une fois la modélisation effectuée, des implémentations sur les différents moteurs seront proposées. Des scripts d'interrogations des bases de données, ainsi que des jeux de données seront également mis à disposition pour pouvoir tester les implémentations.

Ce document accompagne et guide le lecteur :

- D'une part, à comprendre la modélisation proposée grâce à une présentation du diagramme UML.
- Et d'autre part, les raisons techniques et fonctionnelles derrière la multitude de choix de modélisation en se référant à chaque fois à la plateforme ELIA.

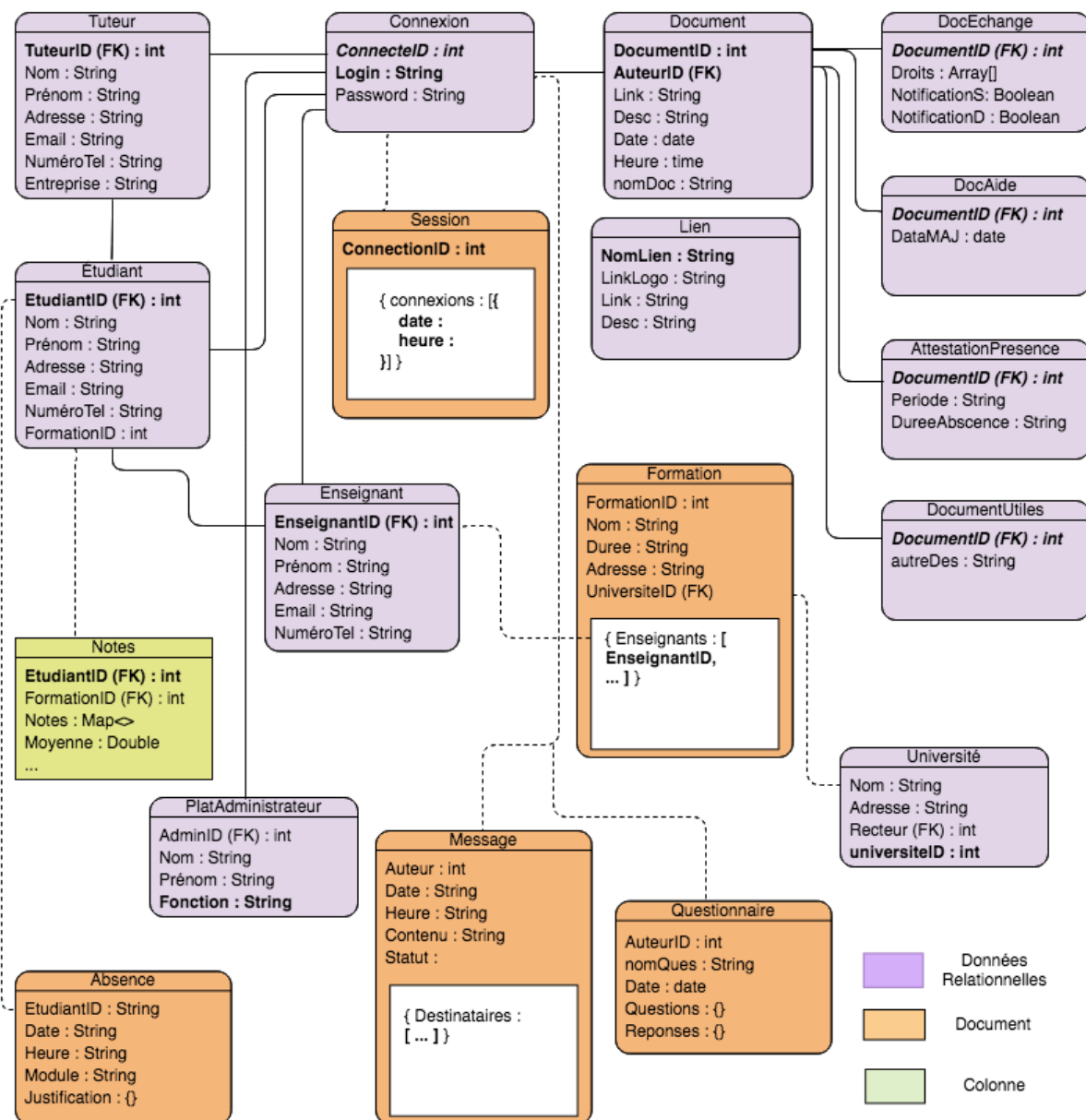
Le code du projet, ainsi que ce rapport sont disponibles sur le dépôt Git à l'adresse : https://github.com/Haouah19/SQL_noSQL_newSQL.

Avant tout développement, il nous paraît important de définir ELIA. Il s'agit d'une plateforme proposée par le CFA AFIA dans le but de centraliser le suivi et les échanges entre les tuteurs entreprises, les apprentis ainsi que les tuteurs enseignants. Ainsi, ELIA accompagne, informe et évalue l'apprenti via un accès internet sécurisé. L'avantage principal de la plateforme est la digitalisation de l'outil d'évaluation. Elle permet un accès aux supports, aux documents et aux liens importants pour que l'évaluation de l'alternant se déroule le plus rapidement et le plus facilement possible.

2. Modélisation du multi-modèle

Voici ci-dessous le multi-modèle de données que nous proposons afin de reproduire l'accès aux données de l'application ELIA.

Les modèles implémentés sont : relationnels, orientés document, ainsi qu'orientés colonne. Nous détaillerons dans les sections suivantes les différentes parties de notre multi-modèle suivant le type de données modélisé.



2.1 Les données relationnelles

2.1.1 Données personnelles

Suivant votre lecteur du modèle ci-dessus, vous remarquerez aisément que toutes les données liées aux personnes ont été stockées dans des tables relationnelles. Le nom, le prénom, l'e-mail - unique - ou l'adresse des étudiants, des tuteurs ou des enseignantes sont des informations importantes qui doivent être cohérentes à tout instant de la vie de l'application. Le fonctionnement de l'application sera perturbé si à un instant une transaction qui modifie l'e-mail par exemple, échoue sans revenir à l'état précédent. La transaction doit être fiable et pérenne. Ces informations doivent respecter les propriétés ACID : Atomicité, cohérence, isolation et durabilité.

Un autre point important nous a guidé dans notre choix, le volume des données. Après réflexion, il est clair que le volume des données des informations personnelles des utilisateurs de l'application n'a nul besoin d'un type de stockage conçu pour des données massives. Aussi, ce sont des données qui ne répondent pas à un besoin d'analyse. Il n'y aura aucune demande d'analyse sur les noms, prénoms ou e-mails. Rien ne justifie le choix d'une base de données noSQL pour stocker ce type de donnée.

Nous avons conçu une table pour représenter les étudiants qui possède comme attribut : le nom, le prénom, l'e-mail et le numéro de téléphone, d'autres attributs peuvent s'ajouter si le besoin métier l'oblige. La table possède aussi un attribut : *FormationID*, ce dernier lie un étudiant à une formation. Ce n'est pas une clé étrangère car les formations sont stockées dans un modèle orienté Document. Il faut que le développeur garantisse au niveau applicatif la cohérence de cette liaison. La table *Étudiant* possède deux clés étrangères qui ne doivent pas être nulles vers les tables *Tuteur* et *Enseignant*. Ces deux clés étrangères représentent le tuteur entreprise, ainsi que le tuteur universitaire de l'étudiant. Ainsi au niveau applicatif, il faut d'abord insérer un enseignant et un tuteur dans la base, puis insérer l'étudiant.

La clé primaire de la table *Étudiant* est un entier qui référence la table *Connexion*, nous reviendrons plus en détails sur ce choix.

La table qui représente les tuteurs possède en plus des attributs de la table *Étudiant* (sauf les clés étrangères et *FormationID*), un attribut *Entreprise*. Un tuteur peut avoir 0 à n étudiants, cette relation a été implémentée grâce à l'attribut *tuteurID* dans la table *Étudiant*. Un enseignant quant à lui, a exactement les mêmes attributs qu'un tuteur (sauf l'attribut *Entreprise*). Les clés primaires des deux tables : *Tuteur* & *Enseignant* sont comme pour la table *Étudiant*, des clés étrangères référençant la table *Connexion*.

Nous avons choisi de stocker les administrateurs du site dans une table nommée *PlatAdministrateur*. Cette table possède une clé primaire *AdminID* qui est une clé étrangère vers la table *Connexion*, ainsi que des attributs tel le nom, le prénom, ...

2.1.2 Logins & mots de passe

Les données de connexion qui se trouvent dans la table *Connexion* doivent elles-aussi respecter les propriétés ACID. Il est important qu'une transaction se fasse au complet ou ne se fasse pas du tout, ainsi on garantit la cohérence des données à tout instant. Les modèles en noSQL n'offrent pas un tel respect des propriétés ACID.

La table *Connexion* stocke les logins et mots de passe des utilisateurs de la base. Les utilisateurs de l'application ne créent pas eux-mêmes leurs comptes. Une fois que le responsable de la formation du CFA enregistre les étudiants, les enseignants, ainsi que les tuteurs, les logins et mots de passe temporaires sont envoyés par mail. L'utilisateur peut ainsi se connecter et changer immédiatement son mot de passe.

On peut résumer cette action techniquement par la séquence suivante : quand l'administrateur insère un nouvel utilisateur, le Back End s'occupe de générer un login et un mot de passe et d'insérer cela dans la table *Connexion*. Puis avec l'identifiant *ConnexionID* retourné, il insère les informations utilisateurs dans la table correspondante avec comme clé primaire, l'identifiant *ConnexionID* retourné. Un e-mail est envoyé à l'utilisateur. Grâce aux informations communiquées dans l'e-mail, l'utilisateur peut se connecter et depuis l'onglet : « Votre fiche personnelle » ajouter et modifier ces informations.

La table *Connexion* possède les attributs suivants :

- *ConnectionID* : C'est un entier généré par le moteur de la base de données. Les tables *Étudiant* (*ÉtudiantID*), *Tuteur* (*TuteurID*), *Enseignant* (*EnseignantID*), *PlatAdministrateur* (*AdminID*) ont comme clé primaire, une référence vers *ConnexionID*. Ainsi, on s'assure l'unicité. On utilise l'attribut *ConnexionID* pour savoir qui est connecté et avoir accès à ses informations.

On aurait pu utiliser des technos Front End (cookie, session, ...) pour savoir à tout instant le login de l'utilisateur connecté à la plateforme mais nous avons préféré utiliser la base de données en vue de l'objectif orienté modèle de ce projet.

- *Login* : Attribut unique, c'est le point d'entrée des connections. Pour que l'accès à cet attribut soit plus efficace, nous avons mis en place un index de type *btree*.
- *Password* : Il ne faut pas insérer le mot de passe en clair. Pour l'implémentation on utilise une extension PostgreSQL (*pgcrypto*) pour hacher le mot de passe dans la base.

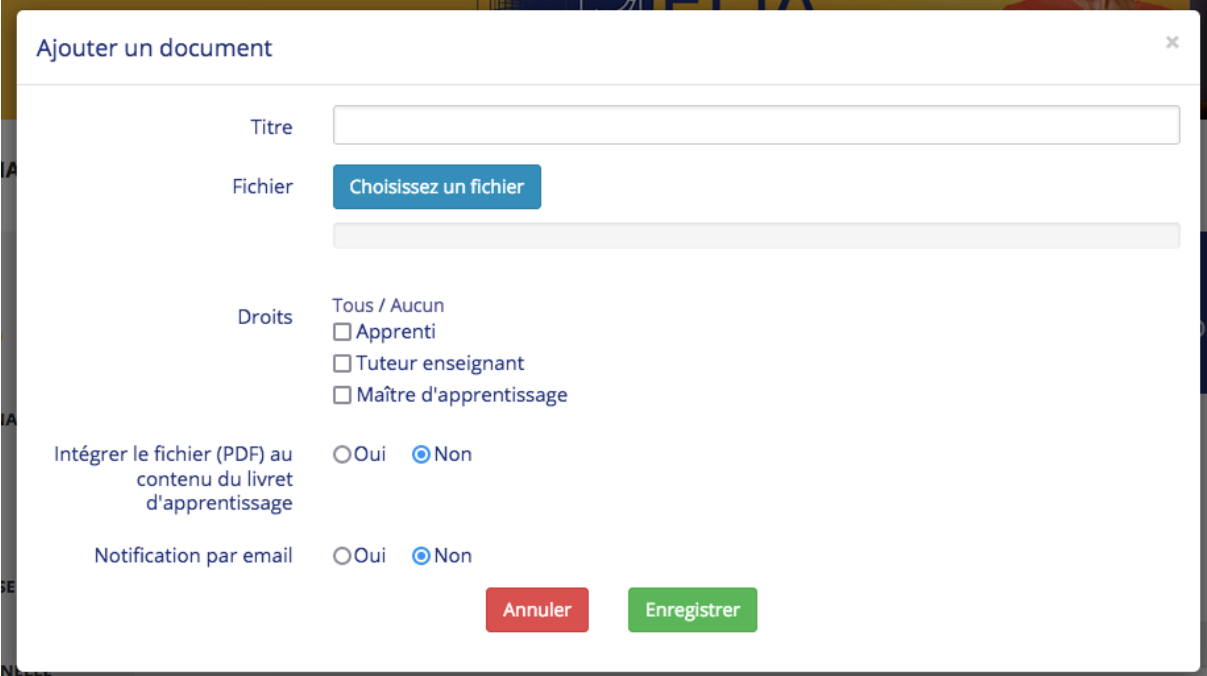
2.1.3 Documents sur la plateforme

Les documents accessibles depuis la plateforme pour un étudiant sont disponibles depuis 4 onglets :

- Échange de docs : Depuis cet espace, le tuteur, l'étudiant ou l'enseignant peuvent s'échanger des documents. Quand un utilisateur décide d'ajouter un document sur la plateforme, il choisit : le titre du document, les droits – Apprenti, Tuteur enseignant ou maître d'apprentissage -, s'il souhaite intégrer le fichier au livret d'apprentissage, si une notification par mail doit être envoyée aux utilisateurs possédant les droits.
- Aide : Des documents à télécharger sont disponibles depuis cet espace. Ils sont accessibles à tous les utilisateurs.
- Attestation de présence : Le responsable du CFA peut charger dans cet espace les attestations de présences de l'étudiant. Cet espace est accessible à chacun.
- Documents utiles : ce sont des documents génériques, accessibles à chacun.

La table *Document* contient les attributs à minima nécessaires pour répondre au besoin d'accès au document sur la plateforme. Elle contient un attribut *DocumentID* qui est la clé primaire de la table. Elle possède également un attribut unique *nomDoc*. Des attributs *Date & Heure* pour stocker la date et l'heure de publication, un attribut *Link* pour stocker le chemin pour accéder au document. La table possède également une clé étrangère nommée *Auteur*, cette clé ne doit pas être nulle. Elle référence la clé primaire de la table *Connexion*. Grâce à cet attribut, on peut retrouver grâce à une jointure les informations de l'auteur tel que le nom, le prénom ou la fonction.

Puis, nous avons conçu 4 tables : *DocEchange*, *DocAide*, *AttestationPresence* et *DocumentUtiles*. Chaque table contient les informations supplémentaires nécessaires à un onglet précis. Chaque table a comme clé primaire une référence vers la clé primaire de la *Document*. C'est ainsi, - comme pour la table *Connexion* – nous garantissons l'unicité.

The image shows a web form titled "Ajouter un document" with a close button (X) in the top right corner. The form contains several fields and options: a text input for "Titre"; a "Fichier" section with a blue button "Choisissez un fichier" and a light gray file list below it; a "Droits" section with a dropdown menu currently showing "Tous / Aucun" and three checkboxes for "Apprenti", "Tuteur enseignant", and "Maître d'apprentissage"; a section for "Intégrer le fichier (PDF) au contenu du livret d'apprentissage" with radio buttons for "Oui" and "Non" (selected); and a "Notification par email" section with radio buttons for "Oui" and "Non" (selected). At the bottom right are two buttons: a red "Annuler" button and a green "Enregistrer" button.

L'image ci-dessus représente l'ajout d'un document dans l'onglet : « Échange de Docs ». Les informations du document seront stockées dans la table *Document*, puis les spécificités dans la table *DocEchange* (tel que les droits, ...). Quand un utilisateur accède à la page, il suffit de charger les informations depuis la table correspondante à l'onglet et de faire une jointure avec la table *Document*. Cette stratégie a été pensée de la sorte, principalement pour éviter la redondance de données dans la base car il y a des documents qui se trouvent dans différents onglets à la fois.

Aussi, le choix s'est porté sur un modèle relationnel car nous pensons que la plateforme répond principalement à un besoin d'échange de document. Ce qui fait que c'est une donnée qui doit être cohérente à tout instant. D'où le relationnel pour les propriétés ACID.

2.1.4 Liens utiles

Pour les liens, la modélisation a été simple car les liens ne sont référencés que dans la l'onglet *Liens utiles*, nous stockons les liens dans une table relationnelle car le volume des données est acceptable. Les données ne sont chargées qu'une fois lors du chargement de la page. Il n'y a pas de concept de droit d'accès ou de modification. L'administrateur du site, insère les liens utiles depuis un espace du site conçu à cet effet et puis, au chargement de la page les liens seront récupérés par une requête SQL simple (sans jointure, ni même d'agrégation). Aussi, ces données n'ont pas besoin d'analyse. La table *Lien* avec ses attributs répond parfaitement aux besoins de la plateforme ELIA.

2.2 Les données orientées documents

2.2.1 Informations de connexion

D'abord, nous allons expliquer pourquoi nous avons modélisé les informations de connexion avec du noSQL au lieu du SQL, puis pourquoi nous avons choisi un modèle orienté document au lieu du modèle orienté clé-valeur ou colonne ou même graphe.

Nous avons décidé de représenter les informations de connexion dans modèle noSQL après avoir pris conscience du volume de données que générerait le stockage des informations à chaque connexion. Nous avons jugé que si à chaque fois qu'un utilisateur se connecte il fallait enregistrer la date et l'heure, le temps de connexion ou même la position géographique, les données deviendront tellement importantes que les stocker et les rechercher dans un modèle relationnel deviendrait moins efficace qu'un modèle noSQL. Un modèle SQL n'est pas scalable. Les modèles de données NoSQL peuvent rendre le processus plus facile. La majorité d'entre eux a été conçue nativement avec des fonctionnalités de « scalabilité ».

Nous aurions pu modéliser cela avec un modèle clé-valeur (Redis) ou par un modèle colonne mais nous avons préféré utiliser un modèle document car :

- **Clé-valeur vs Document :**

La critique qu'on peut faire sur la représentation des informations de connexion par un modèle clé-valeur est le fait que dans modèle clé-valeur, les valeurs sont « opaques ». On ne peut requêter que la clé et pas les valeurs. Ceci amoindrit l'intérêt de stocker les données de connexion.

Nous voulons pouvoir récupérer par exemple les connexions d'un utilisateur entre deux dates données pour valider par exemple que l'utilisateur a pu voir un message que lui a envoyé son tuteur enseignant. Les défenseurs du modèle clé-valeur argumentent en précisant que le modèle clé-valeur répond à ce besoin. Nous leur rétorquons que ce besoin ne peut être fait par un modèle Clé-valeur qu'avec l'ajout d'une couche applicative qui une fois la valeur récupérée, on parse

pour récupérer l'information. Le modèle orienté document nous permet justement d'agréger les valeurs et de ne récupérer que ce qui nous intéresse.

Aussi, avec l'utilisation d'un modèle orienté document, nous sommes libres d'ajouter tous les types de données qu'on souhaite. Par exemple, si on souhaite ajouter des données géographiques, il suffira d'insérer en plus de la date et l'heure de connexion, la position géographique sous la forme (JSON, liste de JSON, ...) que l'on souhaite. Puis, si l'administrateur de la plateforme souhaite visualiser depuis quel emplacement l'accès se fait le plus, une simple requête suffira. Tout ceci est plus compliqué à faire avec un modèle clé-valeur.

- **Colonne vs Document :**

Le choix d'un modèle orienté colonne n'est pas opportun car les informations qu'on souhaite stocker sont relativement identiques pour chaque clé. Pour chaque utilisateur, on souhaite avoir la liste de ses connexions. Un modèle orienté colonne ne peut être conçu pour répondre à ce besoin.

L'_id de la session n'est autre que l'attribut *ConnexionID* de la table *Connexion*. L'objectif derrière ce choix est de retrouver facilement les informations de connexion d'un utilisateur. Ainsi si on souhaite savoir quelles sont les dates et heures de connexion d'un utilisateur avec son nom par exemple, il suffit de récupérer depuis la table *Connexion* son identifiant puis de faire une requête dans *Session* avec l'identifiant comme clé. Le document peut contenir les informations qu'on souhaite sous format JSON. *Connexions* sous forme d'une liste de JSON qui contiennent les dates et heures des connexions. À chaque nouvelle connexion, on ajoute un JSON avec les informations qu'on souhaite dans cette liste de connexions.

2.2.2 Messagerie

Le choix d'un modèle orienté document pour répondre au besoin de stockage des messages est évident. La raison principale est que le nombre et le volume des messages peuvent accroître d'une manière exponentielle. Un modèle noSQL est plus performant pour le stockage, la recherche et l'indexation pour des données de type textuelle.

Côté mise en pratique, pour le cas de la messagerie, on ne s'intéresse pas à la clé mais plutôt aux valeurs qu'on ajoute aux documents.

Quand un utilisateur souhaite envoyer un message il se dirige vers l'onglet *Ma messagerie*. Puis, il peut accéder à un formulaire qui lui permet d'écrire le contenu de son message, de préciser l'objet, d'ajouter des destinataires, de joindre des fichiers et décider des notifications. Toutes ces informations seront stockées dans un document, c'est-à-dire, l'auteur, une liste de destinataires, le contenu, les liens vers les fichiers, date, heure, notification, ... En plus de cela, on ajoute un flag : Archivé. Ce dernier permettra de savoir si un message est archivé ou non. Quand un utilisateur vérifie ses messages, l'application va aller requêter la base pour rechercher tous les messages où figure l'utilisateur connecté dans la liste des destinataires. L'application affichera le résultat.

Une fonctionnalité de réponse sera également ajoutée, quand le destinataire cliquera pour répondre à un message. L'application devra rechercher le message et ajouter la réponse en tant que sous-objet. Chaque message aura ainsi une liste de réponse. Aussi, si l'expéditeur a coché la case : « Recevoir une notification des réponses par email », l'accès aux informations de l'expéditeur sera plus simple.

2.2.3 Absences des étudiants

Nous avons choisi de stocker les données liées à l'absence des étudiants dans un modèle orienté document car :

- Les absences sont par définition des données volumineuses. Chaque étudiant peut s'absenter une multitude de fois.
- Les absences sont des données qui ont besoin d'être manipulées. On doit les requêter pour trouver facilement l'information qu'on recherche. Ce sont des données qui doivent être facilement agrégées. Il n'est pas impossible qu'un maître d'apprentissage souhaite être informé du nombre d'heures d'absence ou des dates d'absence de son étudiant.

En ce qui concerne la modélisation. On représente une absence par un document. L'_id est l'identifiant de l'étudiant. Ceci doit être garanti au niveau applicatif. Le document contient la date, l'heure, ainsi que l'UE. Le document contient aussi un flag qui nous informe si l'étudiant a justifié son absence.

Quand un étudiant souhaite justifier son absence, il insère une justification (représenté dans le document par le lien vers le fichier). Si sa justification est valable, alors le flag devient positif.

2.2.4 Formations

Les informations liées aux formations dispensées par le CFA AFIA sur la plateforme ELIA ont été représentées par un modèle orienté document. Nous pensons que l'objectif réel de l'application n'est pas de rechercher l'exactitude des informations sur les formations ou sur les enseignants mais plutôt de partager des messages, des documents et des supports entre les étudiants, les enseignants, ainsi que les tuteurs. En tant qu'alternant au sein du CFA AFIA, l'application nous aura servi que dans cet objectif. Dans notre choix de modélisation, la priorité est donnée à une modélisation qui permet d'avoir un système de représentation simple des données, évolutive et qui permet de requêter simplement. On s'éloigne du respect des propriétés ACID pour les formations, cela n'est point l'objectif de la plateforme.

Chaque formation contient un identifiant, c'est la clé. Le document contient une référence vers l'université (table *Université*), une liste d'enseignants (table *Enseignant*), un enseignant

responsable, une adresse postale, ... Toute l'intégrité des données et leurs cohérences doit être gérée au niveau applicatif.

Cette représentation nous permet de nous éloigner des jointures lourdes et coûteuses du modèle relationnel, tout en garantissant, une recherche et une agrégation efficace.

2.2.5 Questionnaires

Les raisons derrière le choix de modélisation des données lié aux questionnaires dans un modèle orienté document rejoignent celles de la partie messagerie.

2.3 Les données orientées colonnes

2.3.1 Notes des étudiants

Le choix d'un modèle de données pour représenter les notes des étudiants a été compliqué car :

- Une note est associée à une UE, ce concept est inexistant dans notre modèle.
- Une UE est associée à une formation. Il y'a des formations qui partagent des UEs.
- Chaque étudiant possède une note finale par UE. Chaque formation contient une multitude d'UE.

D'abord, on a voulu représenter ces données par un modèle orienté graphe, où on représente chaque étudiant et chaque UE par un nœud, ainsi qu'une relation « a obtenu » qui représente la note. Cette entreprise n'a pas abouti.

Puis, on s'est tourné vers un modèle orienté document où on stocke dans un document JSON, le nom de chaque UE, ainsi que le note ; avec comme clé l'identifiant de l'étudiant. Une fois cette étape faite, on s'est rendu compte qu'en moyenne un étudiant n'avait que 10 notes car il n'étudie que 10 UEs en moyenne par formation. Il y'a 85 formations au sein du CFA AFIA. Ceci est totalement adapté pour un modèle orienté colonne car si on représente chaque module par une colonne, et l'identifiant de l'étudiant comme clé. Chaque clé n'aura en moyenne que 10 valeurs et chaque colonne n'aura en moyenne que 45 valeurs (car il y a en moyenne 30 étudiants par formation).

id	Module #1 (IF)
Etu IF #1	12
Etu IF #2	13

id	Module (IF-ID)
Etu IF #1	14
Etu IF #2	15
Etu ID #1	13
Etu ID #2	10

id	Module #2 (ID)
Etu ID #1	7
Etu ID #2	8

Le bémol est qu'avec cette représentation (ci-dessus), on aura 850 colonnes pour prendre en charge toutes les UEs de toutes les formations du CFA AFIA. C'est un souci majeur si le besoin évolue. Cette représentation possède aussi des lacunes si on souhaite ajouter plus d'informations à un étudiant tel que la moyenne, ... Il faudra ajouter une nouvelle colonne. En

contrepartie, cette représentation nous permet d'avoir rapidement la note d'un étudiant, ainsi que des statistiques très poussées sur les notes des étudiants.

Alors, pour rester dans un modèle orienté colonne, nous avons conçu une table qui a comme clé primaire l'identifiant de l'étudiant, l'identifiant de la formation, ainsi que le semestre, puis une colonne notes qui est une MAP qui lie une UE à une note. Ainsi, on retrouve les notes de chaque étudiant facilement. On a aussi ajouté une colonne boolean qui nous indique si l'étudiant a validé son semestre et une autre pour représenter la moyenne.

3. Implémentation du multi-modèle

3.1 Les données relationnelles

Afin d'implémenter les modèles, nous avons décidé d'utiliser PostgreSQL car nous avons tous deux une expérience avec ce SGBD. Chacun a téléchargé la dernière version de PostgreSQL en local. M. MOUHOUBI qui s'est occupé de l'implémentation du modèle relationnel a aussi utilisé Postico qui est un client PostgreSQL. Postico facilite l'implémentation en guidant dans les choix des développeurs.

Le fichier **creationTable.sql** joint à ce rapport dans le dossier *implementation/postgreSQL* contient toutes les définitions des tables et des indexes utiles.

Voici un exemple de création de la table *Étudiant* :

```
-- Table Definition -----
CREATE TABLE "Etudiant" (
  "etudiantID" integer PRIMARY KEY REFERENCES "Connexion"("connexionID") ON
DELETE CASCADE ON UPDATE CASCADE,
  "nomEtu" text NOT NULL,
  "prenomEtu" text NOT NULL,
  "adresseEtu" text,
  "emailEtu" text NOT NULL,
  "numeroTelEtu" text NOT NULL,
  "tuteurID" integer REFERENCES "Tuteur"("tuteurID"),
  "enseignantID" integer REFERENCES "Enseignant"("enseignantID"),
  "formationID" integer NOT NULL
);

-- Indices -----
CREATE UNIQUE INDEX "Etudiant_pkey" ON "Etudiant"("etudiantID" int4_ops);
```

Le fichier **AjoutDonnees.sql** – dans le même répertoire – contient les instructions qui permettent d'ajouter un jeu de données à la base de données.

Exemple d'ajout d'un nouvel utilisateur dans la table *Connexion* :

```
INSERT INTO "Connexion" ("connexionID", login, password) VALUES (
  1,
```

```
'ahmed.haouili',
crypt('ahmed', gen_salt('bf'))
);
```

Le fichier **requetes.sql** quant à lui, contient un ensemble de requêtes. Les requêtes simulent un scénario d'exécution.

Voici une requête qui permet de récupérer tous le nom et prénom des étudiants qui ont M. Colazoo comme tuteur enseignant :

```
SELECT "nomEtu", "prenomEtu"
FROM "Etudiant", "Enseignant"
WHERE "Etudiant"."enseignantID" = "Enseignant"."enseignantID"
AND "Enseignant"."nomEn" = 'Colazoo';
```

3.2 Les données orientées document

Sur la partie orientée document, nous avons utilisé le SGBD mongoDB. Nous avons jugé qu'il était plus simple d'utiliser une version en ligne, au lieu de l'avoir en local. Pour cela nous avons utilisé la version gratuite du site mongoDB qui nous permet de créer un cluster avec des performances suffisantes (512 MB de stockage, RAM partagé, ...). Pour accéder au cluster, deux solutions existent :

- Utiliser une connexion mongo Shell qui nous lie au cluster. Cela se fait en installant mongodb-community-shell sur sa machine.
- Utiliser MongoDB Compass qui est La GUI (interface graphique utilisateur) pour MongoDB.

Nous avons opté pour la seconde solution. Nous avons créé les 5 collections modélisées : Absence, Formation, Message, Questionnaire et Session. Les éléments présents dans les documents et les clés des différentes collections ont été discutés dans la partie modélisation.

Les documents sous formats JSON de chacun des collections sont présents dans les fichiers **<nomCollection>.json** dans le répertoire : *implementation/mongoDB*.

```
{
  "_id": "3",
  "nom": "Informatique pour la finance",
  "duree": "1 an",
  "adresse": "Place du Maréchal de Lattre de Tassigny, 75016 Paris",
  "universiteID": {
    "$numberLong": "1"
  },
  "enseignants": [{
    "$numberLong": "5"
  }, {
    "$numberLong": "8"
  }, {
    "$numberLong": "7"
  }],
  "StatutApprentissage": true,
  "niveau": "Bac+5"
}
```

Le document ci-dessus représente l'enregistrement de la formation M2 Informatique pour la finance en Apprentissage à Dauphine. Quant au document ci-dessous représente les connexions de l'utilisateur avec l'identifiant 1.

```
{
  "_id": 1,
  "connexions": [{
    "$date": "2021-06-27T11:21:55.000Z"
  }, {
    "$date": "2021-06-25T13:19:50.000Z"
  }, {
    "$date": "2021-06-23T13:19:50.000Z"
  }]
}
```

Si on souhaite par exemple, savoir quels sont les utilisateurs qui se sont connecté entre le 23/06 et le 27/06, on utilise la requête suivante :

```
{connexions : { $elemMatch: { $gt: new Date('2021-06-23'), $lt: new Date('2021-06-27')} }}
```

Toutes les requêtes mongoDB se trouvent dans le fichier **requests.txt** dans *implementation/mongoDB*.

3.3 Les données orientées colonne

Le SGBD Cassandra a été utilisé pour implémenter la partie du modèle orienté colonne. Depuis le site de datastax (<https://www.datastax.com/try-it-out>) nous avons pu créer un KeySpace - sans nous soucier de l'architecture - grâce à la commande suivante :

```
CREATE KEYSPACE elia WITH replication = {'class': 'SimpleStrategy', 'replication_factor': 1};
```

Puis, nous avons pu créer notre table Notes :

```
CREATE TABLE elia.etudiant_notes (etudiantID INT, formationID INT, semestre INT, moyenne double, admis boolean, notes map<text,double>, PRIMARY KEY (etudiantID, formationID, semestre) );
```

Et y insérer des éléments :

--- Insertion des données ---

```
INSERT INTO elia.etudiant_notes (etudiantID, formationID, semestre, moyenne, admis, notes) VALUES (1,1, 1, 15.16, true,{'Algorithmique' : 7, 'Actuariat': 20, 'Finance des marchés' : 18.5});
```

etudiantid	formationid	semestre	admis	moyenne	notes
1	1	1	True	15.16	{'Actuariat': 20, 'Algorithmique': 7, 'Finance des marchés': 18.5}
1	1	2	True	14	{'Anglais': 16, 'C++': 12, 'microServices': 14}
2	1	1	True	16.33	{'Actuariat': 19, 'Algorithmique': 12, 'Finance des marchés': 18}
2	1	2	True	14.12	{'Anglais': 15, 'C++': 15, 'microServices': 12}

Si on souhaite par exemple avoir accès à la note de C++ d'un étudiant donné, la requête suivante le permet :

```
cqlsh> SELECT notes['C++'] FROM elia.etudiant_notes WHERE etudiantid=1 and semestre=2 and formationid=1;
notes['C++']
-----
12
```

4. Conclusion

La réalisation de ce projet nous aura permis de modéliser un multi-modèle sur un cas concret, réfléchir à une implémentation et essayer de trouver des solutions aux problèmes rencontrés. Il nous aura permis d'acquérir un savoir-faire en plus des connaissances théoriques vues lors de nos cours et TPs.

Nous avons essayé de collaborer de manière sérieuse et professionnelle tout au long du projet, respectant une méthodologie de travail et essayant de se répartir les tâches le plus équitablement possible et de les paralléliser autant que faire se peut. Malheureusement, à cause d'un planning chargé - rendu de la première partie du mémoire et divers projets (C++, Management des organisations, Composants, ...) -, nous n'avons pas eu le temps nécessaire pour implémenter tous nos souhaits. Notamment, un modèle orienté graphe. Tous deux sommes intéressés par ce type de modélisation et de tout ce qu'on peut en tirer comme avantage.

Globalement, nous ressortons de ce projet satisfait de notre travail. Les enseignements acquis durant cette période nous seront sûrement utiles dans le futur, nous essayerons à l'avenir d'accorder plus d'importance à la phase de conception d'un projet, à veiller à lui donner le temps et la réflexion nécessaires afin d'anticiper les éventuels problèmes d'un choix donnés.

Aussi, nous aimerions partager un point que nous trouvons important. Au cours de nos années d'études en informatique, nous n'avons jamais pu prendre part à un module qui nous laisse autant le choix sur les technologies à utiliser pour un projet. On pense que cette manière de faire est très intéressante car elle est très proche de ce qui se fait dans les entreprises. C'est grâce à ce point que le projet a été très intéressant et enrichissant pour nous.