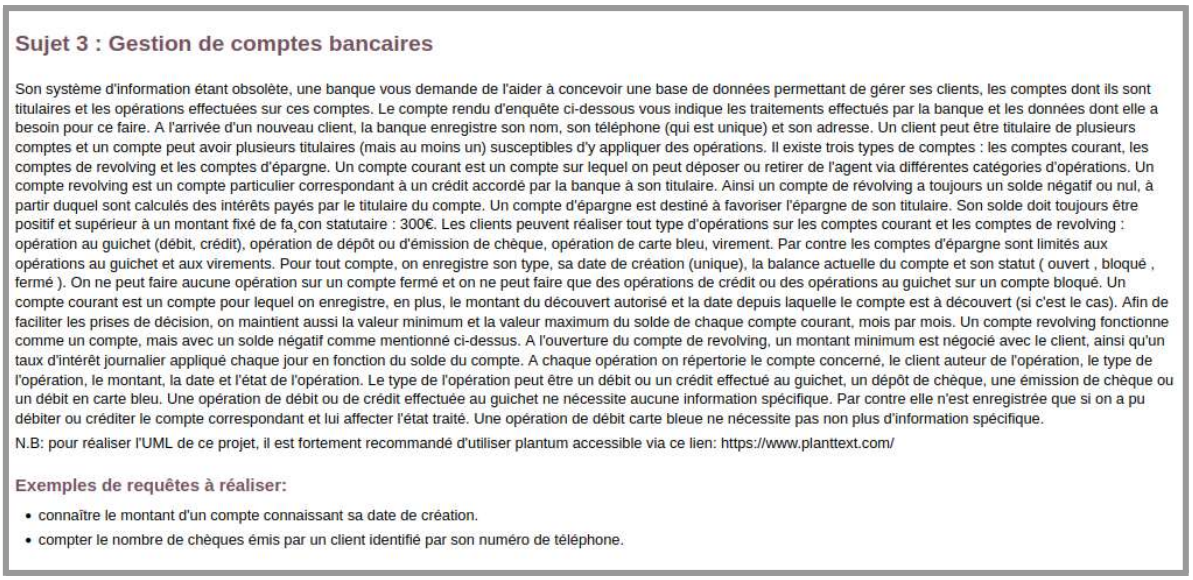
NF18 - Rapport

Sujet 3 : Gestion de comptes bancaires

INTRODUCTION

Dans le cadre de l'UV NF18, nous avons réalisé un projet nous faisant utiliser les notions sur les bases de données apprises au cours du semestre. Ces notions concernent le modèle conceptuel de données (MCD), en UML, le modèle logique de données (MLD), en algèbre relationnelle et l'implémentation, en SQL. Nous avons dans un premier temps fait une implémentation en relationnel avec PostgreSQL, puis en non relationnel, toujours avec PostgreSQL.

En ce qui concerne le sujet du projet en lui-même, nous devions réaliser un système de gestion de comptes bancaires. Le sujet présentait un certain nombre de contraintes et de spécifications, en voici l'énoncé :



Sujet 3 : Gestion de comptes bancaires

Son système d'information étant obsolète, une banque vous demande de l'aider à concevoir une base de données permettant de gérer ses clients, les comptes dont ils sont titulaires et les opérations effectuées sur ces comptes. Le compte rendu d'enquête ci-dessous vous indique les traitements effectués par la banque et les données dont elle a besoin pour ce faire. A l'arrivée d'un nouveau client, la banque enregistre son nom, son téléphone (qui est unique) et son adresse. Un client peut être titulaire de plusieurs comptes et un compte peut avoir plusieurs titulaires (mais au moins un) susceptibles d'y appliquer des opérations. Il existe trois types de comptes : les comptes courant, les comptes de revolving et les comptes d'épargne. Un compte courant est un compte sur lequel on peut déposer ou retirer de l'agent via différentes catégories d'opérations. Un compte revolving est un compte particulier correspondant à un crédit accordé par la banque à son titulaire. Ainsi un compte de révolving a toujours un solde négatif ou nul, à partir duquel sont calculés des intérêts payés par le titulaire du compte. Un compte d'épargne est destiné à favoriser l'épargne de son titulaire. Son solde doit toujours être positif et supérieur à un montant fixé de façon statutaire : 300€. Les clients peuvent réaliser tout type d'opérations sur les comptes courant et les comptes de revolving : opération au guichet (débit, crédit), opération de dépôt ou d'émission de chèque, opération de carte bleu, virement. Par contre les comptes d'épargne sont limités aux opérations au guichet et aux virements. Pour tout compte, on enregistre son type, sa date de création (unique), la balance actuelle du compte et son statut (ouvert , bloqué , fermé). On ne peut faire aucune opération sur un compte fermé et on ne peut faire que des opérations de crédit ou des opérations au guichet sur un compte bloqué. Un compte courant est un compte pour lequel on enregistre, en plus, le montant du découvert autorisé et la date depuis laquelle le compte est à découvert (si c'est le cas). Afin de faciliter les prises de décision, on maintient aussi la valeur minimum et la valeur maximum du solde de chaque compte courant, mois par mois. Un compte revolving fonctionne comme un compte, mais avec un solde négatif comme mentionné ci-dessus. A l'ouverture du compte de revolving, un montant minimum est négocié avec le client, ainsi qu'un taux d'intérêt journalier appliqué chaque jour en fonction du solde du compte. A chaque opération on répertorie le compte concerné, le client auteur de l'opération, le type de l'opération, le montant, la date et l'état de l'opération. Le type de l'opération peut être un débit ou un crédit effectué au guichet, un dépôt de chèque, une émission de chèque ou un débit en carte bleu. Une opération de débit ou de crédit effectuée au guichet ne nécessite aucune information spécifique. Par contre elle n'est enregistrée que si on a pu débiter ou créditer le compte correspondant et lui affecter l'état traité. Une opération de débit carte bleue ne nécessite pas non plus d'information spécifique.

N.B: pour réaliser l'UML de ce projet, il est fortement recommandé d'utiliser plantum accessible via ce lien: https://www.planttext.com/

Exemples de requêtes à réaliser:

- connaître le montant d'un compte connaissant sa date de création.
- compter le nombre de chèques émis par un client identifié par son numéro de téléphone.

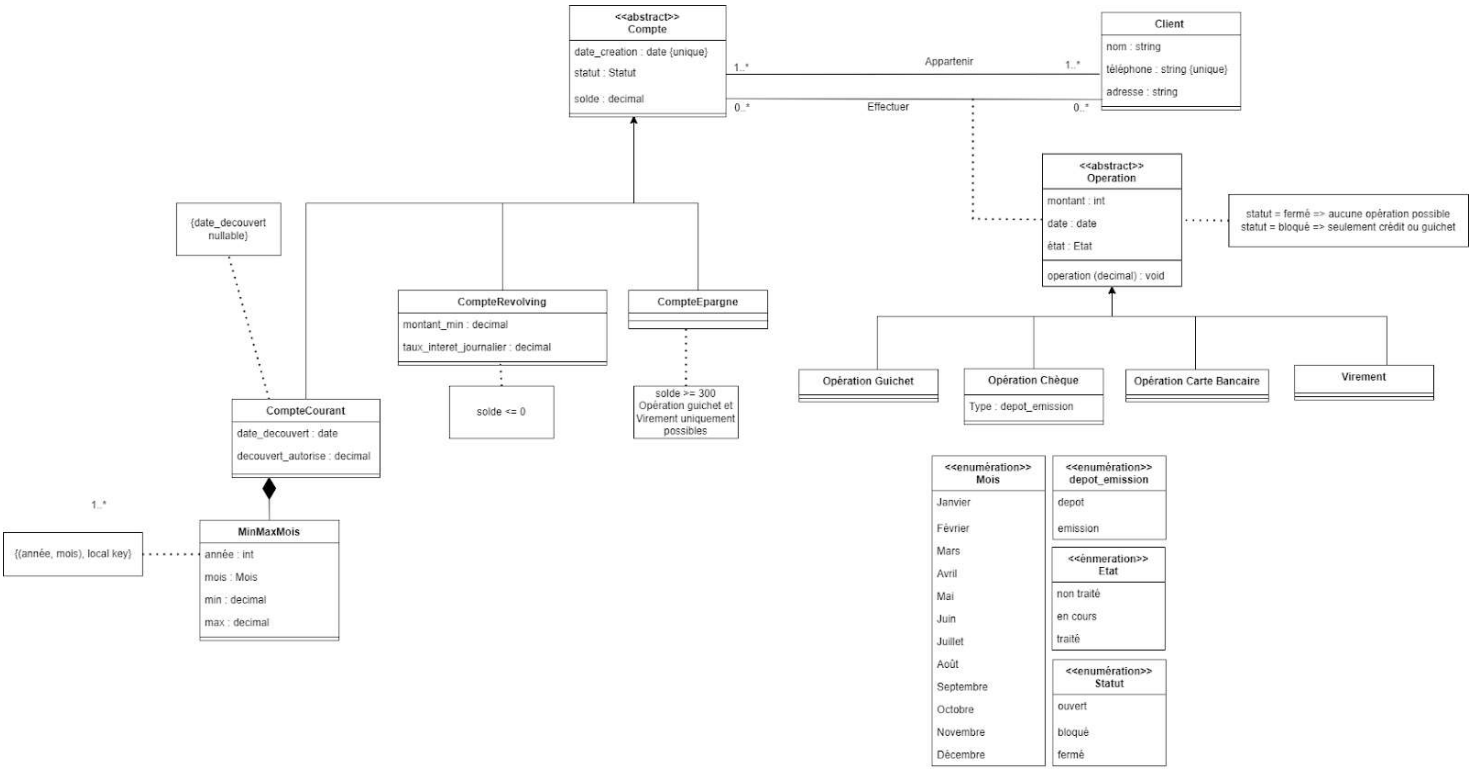
Ce rapport présentera notre démarche pour mener le projet à bien, les difficultés que nous avons rencontrées, les solutions que nous avons trouvées et une comparaison entre les deux implémentations.

Plan

1. Phase 1 : Relationnel
    a. Conception du MCD
    b. Passage au modèle logique de données
    c. Etude de normalisation
2. Phase 2 : Non relationnel
    a. Création des tables
    b. Requêtes
3. Comparaison

Phase I : Relationnel

➢ Conception de l'UML



Nous avons réalisé le MCD de notre projet afin de respecter les consignes données.
Justification de nos choix :
Un Compte peut être de 3 types différents :

- CompteCourant
- CompteRevolving
- CompteEpargne

Ces types de compte héritent donc de la classe Compte puisqu'ils en sont des spécifications.

Les CompteCourant ont une spécificité qui est que le stockage des montants minimum et maximum autorisés du compte pour chaque mois. Nous avons donc introduit la table MinMaxMois qui compose CompteCourant. Chaque tuple va ainsi stocker les différents éléments voulus sur lesquels on pourra faire des vérifications lors du mois courant lorsqu'on voudra mettre à jour le solde avec une opération.

Un compte est lié à un client à la fois par sa possession, mais aussi par les opérations qu'il effectue dessus. La classe Association Opération est héritée par tous les différents types d'opérations. Elles ont toutes leurs contraintes, notamment par rapport aux comptes sur lesquelles elles peuvent être appliquées.

Ainsi, l'ensemble des informations et contraintes demandées peuvent être gérées. Le passage au MLD spécifiera ces dernières.

➢ Passage au Modèle Logique de Données (MLD)

Classes:

Client(#id : int, nom : str, telephone : str, adresse : str) avec {telephone key, (telephone, adresse) NOT NULL}

Compte(#date_creation : date, statut : Statut, solde : decimal) avec {statut, solde NOT NULL}

Appartenir(#compte : => Compte.date_creation, #client => Client.id)

CompteCourant(#compte => Compte.date_creation, date_decouvert : date, decouvert_autorise : decimal) avec {decouvert_autorise NOT NULL}

CompteRevolving(#compte => Compte.date_creation, montant_min : decimal, taux_interet_journalier : decimal) avec {(montant_min, taux_interet_journalier) NOT NULL}

CompteEpargne(#compte => Compte.date_creation)

MinMaxMois (#année : int, #mois : Mois, min : int, max : int, #compte => Compte.date_creation) avec {(min, max) NOT NULL}

Opération(#Client=>Client.id, #Compte=>Compte.date_creation, montant : int, #date : date, état : Etat, type_operation : TypeOpération, TypeChèque : DepotEmission) avec {(Client, montant, état, TypeOpération) NOT NULL}

- **Justification des héritages :**

Pour la classe Opération, on effectue un héritage par classe mère car on est dans le cas où la classe mère est abstraite et où les classes filles ne sont pas référencées par d'autres classes.

Pour la classe Compte, on effectue un héritage par référence. En effet, en raison des relations qui nécessitent un référencement des classes mères et des classes filles, il est impossible d'envisager un héritage par classe mère ou par classe fille sans perdre d'informations.

Types :

DepotEmission : enum{depot, emission}
TypeOpération : enum{Guichet, Chèque, CB, Virement}
Statut : enum{ouvert, fermé, bloqué}
Mois : enum{'janvier', 'février', 'mars', 'avril', 'mai', 'juin', 'juillet', 'août', 'septembre', 'octobre', 'novembre' et 'décembre'}
Etat : enum{'traité', 'non traité', 'en cours'}

Contraintes :

- **Opération :**

NOT (TypeChèque != NULL AND TypeOpération IN {Guichet, CB , Virement})
NOT (Compte.Statut = fermé)
NOT (Compte.Statut = bloqué AND TypeOpération IN {CB, Chèque})
NOT (montant < 0 AND TypeChèque = {dépôt})
NOT (montant > 0 AND TypeChèque = {émission})
NOT (montant > 0 AND TypeOpération IN {Virement, CB})

- **Comptes :**

Compte épargne :
solde > 300 :
Projection(Jointure(Restriction(Compte,solde >=300),CompteEpargne,Compte.date_creation = CompteEpargne.compte), Compte.date_creation) = Projection(CompteEpargne, compte)

{seules des opérations au guichet et virements peuvent faits avec un compte épargne}

Compte revolving :
solde < 0
Projection(Jointure(Restriction(Compte, solde < 0),CompteRevolving,Compte.date_creation = CompteRevolving.compte), Compte.date_creation) = Projection(CompteRevolving, compte)

Classe mère abstraite :
Projection(Compte, id) = Union(Projection(CompteEpargne, compte), (Union(Projection(CompteRevolving, compte), Projection(CompteCourant, compte)))

Héritage exclusif :
Intersection(Projection(Comptecourant, compte), Intersection(Projection(CompteRevolving, compte), Projection(CompteEpargne, compte)))={}

Pour la composition :
Projection(CompteCourant, compte) = Projection(MinMaxMois, compte)

- **Appartenir :**

Tous les clients ont au moins un client et un client possède au moins un compte :

Projection(Restriction(Appartenir, compte)) = Projection(Compte, date_creation)
Projection(Restriction(Appartenir, client)) = Projection(Client, id)

➢ Étude de normalisation

Nous avons ensuite étudié la normalisation fonctionnelle de notre modèle logique initial.

Rappel : 1 NF : Attribut atomique et clé 2 NF : Tout attribut non clé n'est pas déterminé par un sous ensemble de la clé 3 NF : Tout attribut non clé n'est pas déterminé par un attribut non clé BCNF : Tout attribut clé n'est pas déterminé par un attribut non clé

- **Client**(#telephone, nom, adresse) avec {(nom, adresse) NOT NULL}

Dépendances fonctionnelles élémentaires :

telephone → nom
telephone → adresse

Niveau de normalisation fonctionnelle :

BCNF : toutes les dépendances fonctionnelles sont de la forme K → A avec K un attribut clé.

- **Compte**(#date_creation, statut, solde)

Dépendances fonctionnelles élémentaires :

date_creation → statut
date_creation → solde

Niveau de normalisation fonctionnelle :

BCNF : toutes les dépendances fonctionnelles sont de la forme K → A avec K un attribut clé.

- **Appartenir**(#compte => Compte.date_creation, #client => Client.id)

Dépendances fonctionnelles élémentaires :

Il n'y a aucune dépendance fonctionnelle élémentaire, toutes les dépendances fonctionnelles ici sont réflexives.

La table ne contient des clés candidates donc que dépendances réflexives triviales.

Niveau de normalisation fonctionnelle :

BCNF : la clé contient tous les attributs de la table, Appartenir est en BCNF de façon triviale.

- **CompteCourant**(#compte => Compte.date_creation, date_decouvert : date, decouvert_autorise : decimal)

Dépendances fonctionnelles élémentaires :

compte → date_decouvert
compte → decouvert_autorise

Niveau de normalisation fonctionnelle :

BCNF : toutes les dépendances fonctionnelles sont de la forme K → A avec K un attribut clé.

- **CompteRevolving**(#compte => Compte.date_creation, montant_min : decimal, taux_interet_journalier : decimal)

Dépendances fonctionnelles élémentaires :

compte → montant_min
compte → taux_interet_journalier

Niveau de normalisation fonctionnelle :

BCNF : la clé ne contenant qu'un attribut unique, CompteRevolving est en BCNF de façon triviale

- **CompteEpargne**(#compte => Compte.date_creation)

Dépendances fonctionnelles élémentaires :

Il n'y a aucune dépendance fonctionnelle élémentaire, toutes les dépendances fonctionnelles ici sont réflexives.

Niveau de normalisation fonctionnelle :

BCNF : la clé contient tous les attributs de la table, CompteEpargne est en BCNF de façon triviale.

- **MinMaxMois** (#année : int, #mois : Mois, min : int, max : int, #compte => Compte.date_creation)

Dépendances fonctionnelles élémentaires :

annee, mois, compte → min, max

Niveau de normalisation fonctionnelle :

BCNF : toutes les dépendances fonctionnelles sont de la forme K → A avec K un attribut clé

- **Opération**(#Client=>Client.id, #Compte=>Compte.date_creation, montant : int, #date : date, état : Etat, type_operation : TypeOpération, TypeChèque : DepotEmission)

Dépendances fonctionnelles élémentaires :

Client, Compte → montant, état, type_opération, TypeChèque

Niveau de normalisation fonctionnelle :

BCNF : toutes les dépendances fonctionnelles sont de la forme K → A avec K un attribut clé

Conclusion : La base de données est en BCNF.
En effet :

- **Toutes les tables sont en 1NF car elles comportent une clé et les attributs sont atomiques.**
- **Toutes les tables sont en 2NF parce que tous les attributs non clé sont déterminés par tous les attributs clé et non une partie seulement des attributs clé.**
- **Toutes les tables sont en 3NF puisqu'aucun attribut non clé ne détermine un autre attribut non clé**
- **Toutes les tables sont en BCNF puisque les dépendances fonctionnelles sont de la forme K → A, avec K une clé.**

Par rapport à notre premier MLD, la seule modification effectuée afin de passer d'un modèle en 3NF à un modèle en BCNF a été de supprimer la clé artificielle de Client pour la remplacer par le numéro de téléphone. Cela n'entraîne pas de modification du MCD effectué dans la première partie du projet.

Phase II : Non relationnel

Nous avons effectué le passage au non relationnel en utilisant du JSON-PSQL.

➢ Création de tables

Nous avons rassemblé toutes les informations dans une unique table compte de la forme suivante :

```
CREATE TABLE Compte(
date_creation date primary key,
statut Statut not null,
solde decimal not null,
proprietaires JSON,
type_compte JSON
);
```

Les attributs JSON propriétaires et type_compte contiennent les informations suivantes :

```
proprietaires
{
        “id” :
        “nom” :
        “prenom” :
        “operations” :
                {
                        “montant” :
                        “date” :
                        “etat” :
                        “typeOperation” :
                }
},
```

On y stocke l’ensemble des informations du client ainsi que toutes ses opérations dans le sous attribut JSON operation, qui ne seront donc pas amenés à évoluer au fur et à mesure du temps, ce qui n’est pas adapté à un usage bancaire comme vu en cours.

```
type_compte
{
        “type” :
        “decouvert_autorise” :
        “MinMaxMois” :
                {
                        “annee” :
                        “mois” :
                        “min” :
                        “max” :
                }
```

}’

On y stocke l’ensemble des informations liées au type de compte. decouvert_autorise et le sous attribut JSON MinMaxMois ne seront renseignés que pour un type de compteCourant.

➢ Insertion de données

Voici un exemple d’insertion de données :

```
INSERT INTO Compte VALUES (
'2022-05-03',
'ouvert',
-650,
‘[
{
        “id” : 4,
        “nom” : "Garcia",
        “prenom” : "Fanny",
        “operations” : [
                {
                        “montant” : 100,
                        “date” : “2022-05-12”,
                        “etat” : “traite”,
                        “typeOperation” : “Virement”
                },
                {
                        “montant” : 150,
                        “date” : “2022-05-13”,
                        “etat” : “traite”,
                        “typeOperation” : “Guichet”
                }
        ]
},
{
        “id” : 3,
        “nom” : "Jean",
        “prenom” : "Nemar",
        “operations” : [
                {
                        “montant” : 600,
                        “date” : “2022-05-18”,
                        “etat” : “traite”,
                        “typeOperation” : “Cheque”,
                        “typeCheque” : “Emission”
                }
        ]
}
```

]’,
‘{
“type” : “CompteRevolving”,
“montant_min” : -1500,
“taux_interet_journalier” : 0.6
}’
)

➢ **Requêtes**

```sql = ‘”
```

Comparaison

En ce qui concerne le relationnel, celui-ci dispose de plusieurs tables liées entre elles et disposant de contraintes. Le système est bien ordonné, il n'a pas de redondance et permet une vérification simple de la validité des données. Les mises à jour sont également simples à effectuer. Une critique que nous pouvons faire des contraintes imposées par le sujet est le fait que la date de création de compte est unique, ce qui limite la banque à ne pouvoir créer qu'un seul compte par jour.

Pour la partie non relationnelle, elle ne contient qu'une seule table : compte, contenant elle-même les autres tables. On nous avait explicitement prévenus en cours que le non relationnel n'était pas conçu pour ce type d'application, et ce, pour de multiples raisons : le non relationnel ne nous permet pas de faire de mise à jour, on ne peut donc pas ajouter de transaction après avoir créé un compte. On doit rentrer les opérations futures d'un client au moment de la création du compte, la consigne n'est donc pas fidèle à la réalité. Une autre raison est la moins grande fiabilité des données : il peut y avoir de la redondance et des données incorrectes ou incomplètes, en effet, il n'y a pas ou peu de vérification au niveau de la base de données. S'il doit y avoir des vérifications, elles doivent se faire au niveau applicatif, ce qui diminue la fiabilité des données. Cependant, un avantage du non relationnel est qu'il est plus rapide étant donné qu'il n'y a pas de jointure à faire pour accéder aux données.

Conclusion

À travers ce projet, nous avons su répondre à la demande du sujet en prenant en compte les différentes contraintes et demandes imposées, et ce grâce aux différentes notions que nous avons vues en cours. Nous nous sommes rendus compte que le relationnel était plus adapté pour la gestion de comptes bancaires : il permet de la fiabilité des données, des vérifications et mises à jour faciles. Il permet également d'éviter les redondances.