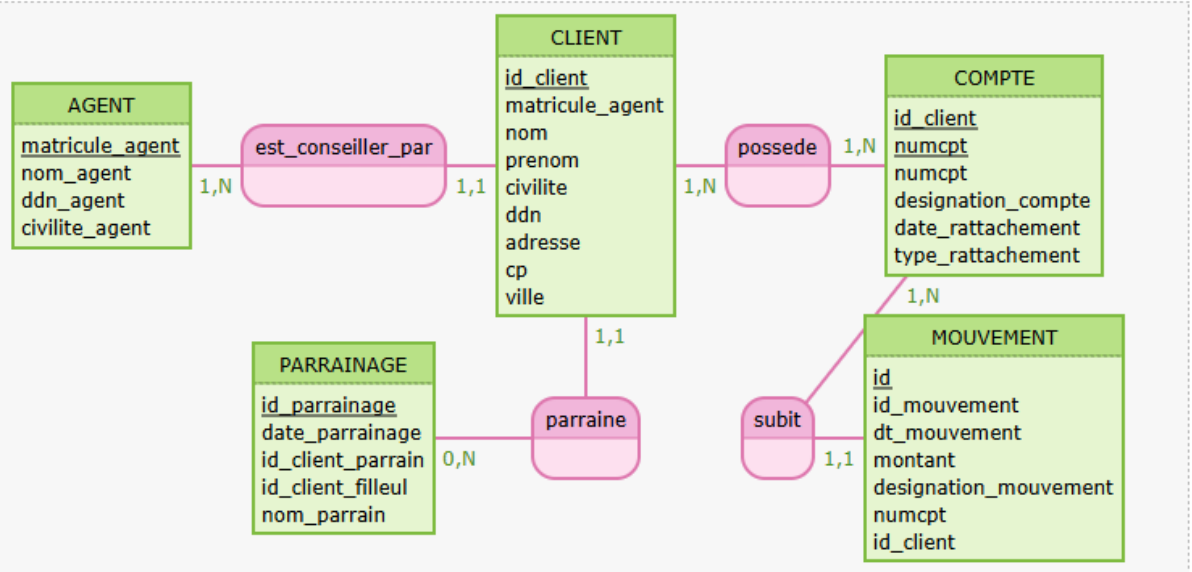

- TOURE Ibrahima

# INTRODUCTION

Dans le cadre de ce projet de SAE 2-01 *Conception et implémentation d'une base de données*, j'ai travaillé à la modernisation du système de gestion des comptes d'une petite banque clermontoise. Jusqu'à présent, cette banque utilisait uniquement un fichier CSV pour gérer les informations de ses clients et de leurs comptes. L'objectif principal de mon travail est donc de créer une SGBD en PostgresSQL plus fiable, plus claire et capable de répondre aux besoins exprimés par le directeur de l'agence.

# I. Modèle Conceptuel des Données :




Dans ce MCD, j'ai identifié cinq entités principales : **AGENT**, **CLIENT**, **COMPTE**, **MOUVEMENT** et **PARRAINAGE**. Ces entités correspondent aux besoins exprimés par le directeur de la banque et permettent de structurer les données de façon claire.

Un agent peut gérer plusieurs clients, mais chaque client n'a qu'un seul agent responsable. Cette organisation permet au client de toujours savoir à qui s'adresser en cas de besoin, c'est une volonté du directeur : le client doit savoir à qui s'adresser.

Un client peut posséder plusieurs comptes, par exemple des comptes communs ou associatifs. De leur côté, les comptes peuvent être liés à plusieurs clients, car un compte commun peut appartenir à plusieurs personnes. Cette relation plusieurs-à-plusieurs est bien prise en compte dans le modèle.

Un compte peut enregistrer plusieurs mouvements, comme des dépôts ou des retraits. Chaque mouvement est lié à un seul compte. Même si, dans le fichier CSV, on remarque que id_mouvement peut apparaître plusieurs fois, je ne saurai dire pourquoi. Normalement, un identifiant (id_mouvement) doit être unique pour chaque mouvement. Sinon, cela peut poser problème pour identifier chaque opération (risque de doublon, d'erreur de suivi, etc.).

Pour gérer les parrainages, la modelisation est faite de sorte q'un client peut parrainer plusieurs autres clients (filleuls). En revanche, un filleul ne peut être parrainé qu'une seule fois. Mais aussi le couple (parrain,filleul) est unique  Cela correspond à la demande de pouvoir savoir qui a parrainé qui et quand, comme le souhaite le directeur.

## II. Schéma relationnels (modèle logique) respectant les règles de normalisations :

**AGENT** ( matricule_agent, nom_agent, ddn_agent, civilite_agent )

- Le champ *matricule_agent* constitue la clé primaire de la table.

**CLIENT** ( id_client, matricule_agent 1, nom, prenom, civilite, ddn, adresse, cp, ville, #matricule_agent 2, #id_parrainage )

- Le champ *id_client* constitue la clé primaire de la table.
- Le champ *matricule_agent 2* est une clé étrangère. Il refernce l'entité *AGENT* en perdant son caractère identifiant.
- Le champ *id_parrainage* est une clé étrangère. Il refernce l'entité *PARRAINAGE* en perdant son caractère identifiant.

**COMPTE** ( id_client, numcpt, numcpt 2, designation_compte, date_rattachement, type_rattachement )

- Les champs *id_client* et *numcpt* constituent la clé primaire composite de la table.

**MOUVEMENT** ( id, id_mouvement, dt_mouvement, montant, designation_mouvement, numcpt 1, id_client 1, #id_client 2, #numcpt 2 )

- Le champ *id* constitue la clé primaire de la table.
- Les champs *id_client 2* et *numcpt 2* sont des clés étrangères. Ils referencent l'entité *COMPTE* en perdant leur caractère identifiant.

**PARRAINAGE** ( id_parrainage, date_parrainage, id_client_parrain, id_client_filleul, nom_parrain )

- Le champ *id_parrainage* constitue la clé primaire de la table.
- les champs id_client_parrain et id_client_filleul constituent des cles étrangères qui referencent id_client de la relation client

### III. Les scripts de création de tables

L'ordre de création des tables suit une logique de dépendance entre les entités. D'abord, on commence par les tables qui ne dépendent d'aucune autre, puis on avance vers celles qui ont des clés étrangères, donc des liens. Le choix d'utiliser une clé primaire composite dans la table compte (formée de id_client et numcpt) permet d'éviter la création d'une table intermédiaire supplémentaire. Cette solution est à la fois simple et efficace, car elle permet de savoir directement à quel client est rattaché un compte, tout en acceptant qu'un client puisse avoir plusieurs comptes (personnels, joints, associatifs). Elle garde aussi une traçabilité claire, en permettant d'identifier qui a fait quoi et quand. De plus, dans la table mouvement, la présence de l'id_client en plus de la référence au compte permet d'avoir une information précieuse pour savoir quel client est concerné par le mouvement, même si le compte est partagé.

-- Création de la table AGENT (sans dépendances)

```
CREATE TABLE agent (
   matricule_agent BIGINT PRIMARY KEY,
   nom_agent VARCHAR(50) NOT NULL,
   ddn_agent DATE,
   civilite_agent VARCHAR(10)
);
```

-- Création de la table CLIENT (dépend de AGENT)

```
CREATE TABLE client (
   id_client BIGINT PRIMARY KEY,
   nom VARCHAR(50) NOT NULL,
   prenom VARCHAR(50) NOT NULL,
   civilite VARCHAR(10),
   ddn DATE,
   adresse TEXT,
   cp VARCHAR(10),
   ville VARCHAR(50),
   matricule_agent BIGINT,
   CONSTRAINT fk_client_agent FOREIGN KEY (matricule_agent) REFERENCES agent(matricule_agent)
);
```

-- Création de la table COMPTE (dépend de CLIENT) avec clé primaire composite

```
CREATE TABLE compte (
   numcpt BIGINT NOT NULL,
   designation_compte VARCHAR(100),
   date_rattachement DATE,
   type_rattachement VARCHAR(50),
   id_client BIGINT NOT NULL,
   CONSTRAINT pk_compte PRIMARY KEY (id_client, numcpt)
);
```

-- création de la table mouvement

```
CREATE TABLE mouvement (
   id SERIAL PRIMARY KEY,
   id_mouvement BIGINT,
   dt_mouvement DATE NOT NULL,
   montant NUMERIC(15,2) NOT NULL,
   designation_mouvement VARCHAR(100),
   numcpt BIGINT NOT NULL,
   id_client BIGINT NOT NULL,
   CONSTRAINT fk_mouvement_compte FOREIGN KEY (numcpt, id_client)
   REFERENCES compte(numcpt, id_client)
);
```

-- Création de la table PARRAINAGE (dépend de CLIENT ×2)

```
CREATE TABLE parrainage (
   id_parrainage SERIAL PRIMARY KEY,
   id_client_parrain BIGINT NOT NULL,
   id_client_filleul BIGINT NOT NULL,
   date_parrainage DATE NOT NULL,
   nom_parrain VARCHAR(100),
   CONSTRAINT unique_couple_parrain_filleul UNIQUE (id_client_parrain, id_client_filleul),
   CONSTRAINT fk_parrain FOREIGN KEY (id_client_parrain) REFERENCES client(id_client),
   CONSTRAINT fk_filleul FOREIGN KEY (id_client_filleul) REFERENCES client(id_client)

);
ALTER TABLE parrainage
ADD CONSTRAINT unique_filleul UNIQUE (id_client_filleul); --un filleul ne peut etre parrainer q'une seule fois
```

## IV. Les scripts d'alimentaion de tables et d'exportation des données

Pour alimenter ma base de données, j'ai choisi une méthode simple et progressive. Après avoir créé les tables dans PostgresSQL, j'ai décidé de ne pas importer directement le fichier brut sae_donnees_bancaires.csv, car il contient trop de colonnes et il est difficile à gérer tel quel. Cela rend l'importation directe presque impossible, surtout si l'on veut identifier rapidement les erreurs.

J'ai donc opté pour une solution plus claire : découper le fichier CSV principal en plusieurs fichiers plus petits, chacun correspondant à une table spécifique de la base de données. Par exemple, pour alimenter la table client, j'ai extrait uniquement les colonnes nécessaires à partir du fichier global, en créant un nouveau fichier CSV nommé client.csv. J'ai fait de même pour les autres tables, Chaque fichier contient uniquement les données utiles pour sa table.

Pour automatiser ce découpage, j'ai utilisé des scripts Python. Grâce à cette méthode, je peux plus facilement contrôler les erreurs, et je sais exactement dans quel fichier (et donc dans quelle table) une modification est nécessaire. C'est aussi une bonne façon de repérer les colonnes problématiques.

En appliquant cette méthode, j'ai découvert que certaines colonnes posaient problème. Par exemple, les colonnes id_mouvement et numcpt ne peuvent pas être utilisées comme clés primaires seules, car elles contiennent des doublons, ce qui viole la contrainte d'unicité imposée par une clé primaire. Il faut donc adapter la structure des tables (comme utiliser une clé primaire composite pour compte).

Un autre problème rencontré concerne les dates. En particulier, certaines lignes du fichier contiennent la date du 29 février pour des années non bissextiles, ce que PostgreSQL ne peut pas accepter. Cela provoque des erreurs à l'importation. Plutôt que de supprimer ces lignes (car cela représenterait une perte de données importante,plus de cinquante lignes), j'ai décidé de corriger ces dates en les remplaçant par le 1er mars. Ainsi, je conserve toutes les informations tout en évitant les erreurs d'insertions.

Enfin, pour garder une trace claire des lignes modifiées, j'ai créé un fichier spécial contenant les id_client et les noms des colonnes (date_rattacment et dt_mouvement) associés à ces erreurs de dates. Cela me permet de retrouver rapidement les clients concernés si besoin, et de vérifier la cohérence des modifications effectuées.

--les scripts d'alimentations

```
\COPY client(id_client, nom, prenom, civilite, ddn, adresse, cp, ville, matricule_agent) FROM 'C:/Users/ibtou/OneDrive/Con-Im-BD/client.csv' DELIMITER ',' CSV HEADER;

\COPY agent(matricule_agent, nom_agent, ddn_agent, civilite_agent) FROM 'C:/Users/ibtou/OneDrive/Con-Im-BD/agent.csv' DELIMITER ',' CSV HEADER;

\COPY compte(numcpt, designation_compte, date_rattachement, type_rattachement, id_client) FROM 'C:/Users/ibtou/OneDrive/Con-Im-BD/compte.csv' DELIMITER ',' CSV HEADER;
```

```
\COPY mouvement(id_mouvement, dt_mouvement, montant, designation_mouvement, numcpt, id_client) FROM
'C:/Users/ibtou/OneDrive/Con-Im-BD/mouvement_modifie.csv' DELIMITER ',' CSV HEADER;

\COPY parrainage(id_client_parrain, id_client_filleul, date_parrainage, nom_parrain) FROM 'C:/Users/ibtou/OneDrive/Con-Im-
BD/parrainage_corrige.csv' DELIMITER ',' CSV HEADER;
```

--les scripts d'exporations

```
\COPY client(id_client, nom, prenom, civilite, ddn, adresse, cp, ville, matricule_agent) TO 'C:/Users/ibtou/OneDrive/Con-Im-
BD/export_client.csv' DELIMITER ',' CSV HEADER;
\COPY agent(matricule_agent, nom_agent, ddn_agent, civilite_agent) TO 'C:/Users/ibtou/OneDrive/Con-Im-BD/export_agent.csv'
DELIMITER ',' CSV HEADER;
\COPY compte(numcpt, designation_compte, date_rattachement, type_rattachement, id_client) TO 'C:/Users/ibtou/OneDrive/Con-Im-
BD/export_compte.csv' DELIMITER ',' CSV HEADER;
\COPY mouvement(id_mouvement, dt_mouvement, montant, designation_mouvement, numcpt, id_client) TO
'C:/Users/ibtou/OneDrive/Con-Im-BD/export_mouvement.csv' DELIMITER ',' CSV HEADER;
\COPY parrainage(id_client_parrain, id_client_filleul, date_parrainage, nom_parrain) TO 'C:/Users/ibtou/OneDrive/Con-Im-
BD/export_parrainage.csv' DELIMITER ',' CSV HEADER;
```

## V. Création de Vues sur les comptes dormants et etat des parrainages :

Dans la première vue comptes_dormants, j'ai choisi de considérer comme "dormant" un compte dont la dernière activité est antérieure au 31 décembre 2021. Ce choix signifie que l'année de référence du projet est 2022. Tous les comptes sans mouvement depuis le 1er janvier 2022 sont donc considérés inactifs.

Dans la seconde vue etat_parrainages, j'ai filtré les enregistrements à partir du 1er janvier 2022, pour ne conserver que les parrainages effectués dans l'année 2022, conformément à la demande du directeur.

Enfin, dans les deux cas, j'ai utilisé la commande \COPY pour permettre au directeur d'exporter les données des vues au format **CSV**, utilisable directement dans un tableur comme Excel.

--Scripts de vues et son export ( comptes dormants)

```
CREATE OR REPLACE VIEW comptes_dormants AS
SELECT
   c.nom AS nom_client,
   c.prenom AS prenom_client,
   cp.numcpt,
   MAX(m.dt_mouvement) AS date_dernier_mouvement,
   SUM(m.montant) AS solde_compte
FROM compte cp
JOIN client c ON cp.id_client = c.id_client
JOIN mouvement m ON m.numcpt = cp.numcpt AND m.id_client = cp.id_client
GROUP BY c.nom, c.prenom, cp.numcpt, cp.id_client
HAVING MAX(m.dt_mouvement) < DATE '2021-12-31';

-- Export CSV
\COPY (SELECT * FROM comptes_dormants) TO 'C:/Users/ibtou/OneDrive/Con-Im-BD/comptes_dormants.csv' CSV HEADER;
```

--Scripts de vues et son export (etat des parrainages)

```
CREATE OR REPLACE VIEW etat_parrainages AS
SELECT
   a.nom_agent,
   p_client.nom AS nom_parrain,
   f_client.nom AS nom_filleul,
```

```
    pr.date_parrainage,
    MAX(m.dt_mouvement) AS date_dernier_mouvement_parrain
FROM parrainage pr
JOIN client p_client ON pr.id_client_parrain = p_client.id_client
JOIN client f_client ON pr.id_client_filleul = f_client.id_client
JOIN agent a ON p_client.matricule_agent = a.matricule_agent
LEFT JOIN mouvement m ON m.id_client = p_client.id_client
WHERE pr.date_parrainage >= DATE '2022-01-01'
GROUP BY a.nom_agent, p_client.nom, f_client.nom, pr.date_parrainage;

-- Export CSV
\COPY (SELECT * FROM etat_parrainages) TO 'C:/Users/ibtou/OneDrive/Con-Im-BD/etat_parrainages.csv' CSV HEADER;
```