Projet Final CS50 SQL – Base de données bancaires

1. Objectif

La base de données a pour objectif de modéliser les opérations fondamentales d'une banque, en mettant l'accent sur la gestion des clients, des comptes, des transactions financières et des succursales. Elle vise à répondre à deux besoins principaux :

- Le suivi en temps réel : il est important d'offrir une vision actualisée et exhaustive des soldes des comptes, des transactions récentes et des informations clients.
- Analyse historique : pouvoir permettre l'audit des activités passés, comme la traçabilité des transactions. La génération de statistiques globales, comme la répartition par types de comptes.

Ce système pourrait servir de fondation pour des applications essentielles au bon fonctionnement d'une institution bancaire. En effet, il est en mesure d'alimenter des portails clients en ligne, en fournissant en temps réel des informations fiables sur les soldes, les opérations récentes ou les produits souscrits. Il soutient également des outils de gestion interne pour les employés, leur permettant de suivre les interactions avec les clients, d'analyser leurs besoins et de proposer des services adaptés. Les rapports réglementaires pour les autorités financières sont facilités car la base de données assure une traçabilité.

En centralisant des données cohérentes, sécurisées et accessibles, ce système pourrait renforcer la qualité du service clients, et la performance opérationnelle d'un établissement.

*Après avoir déterminé les objectifs généraux de la base de données, il faut clairement définir ses fonctionnalités. Le périmètre décrit ici explique les éléments principaux du système, en les reliant aux objectifs initiaux. Cela permet de s'assurer que le modèle reflète de manière logique et pratique les opérations d'une banque, pour qu'il soit facile à utiliser au quotidien.*

2. Portée

La base de données couvre trois domaines clés qui sont les suivants :

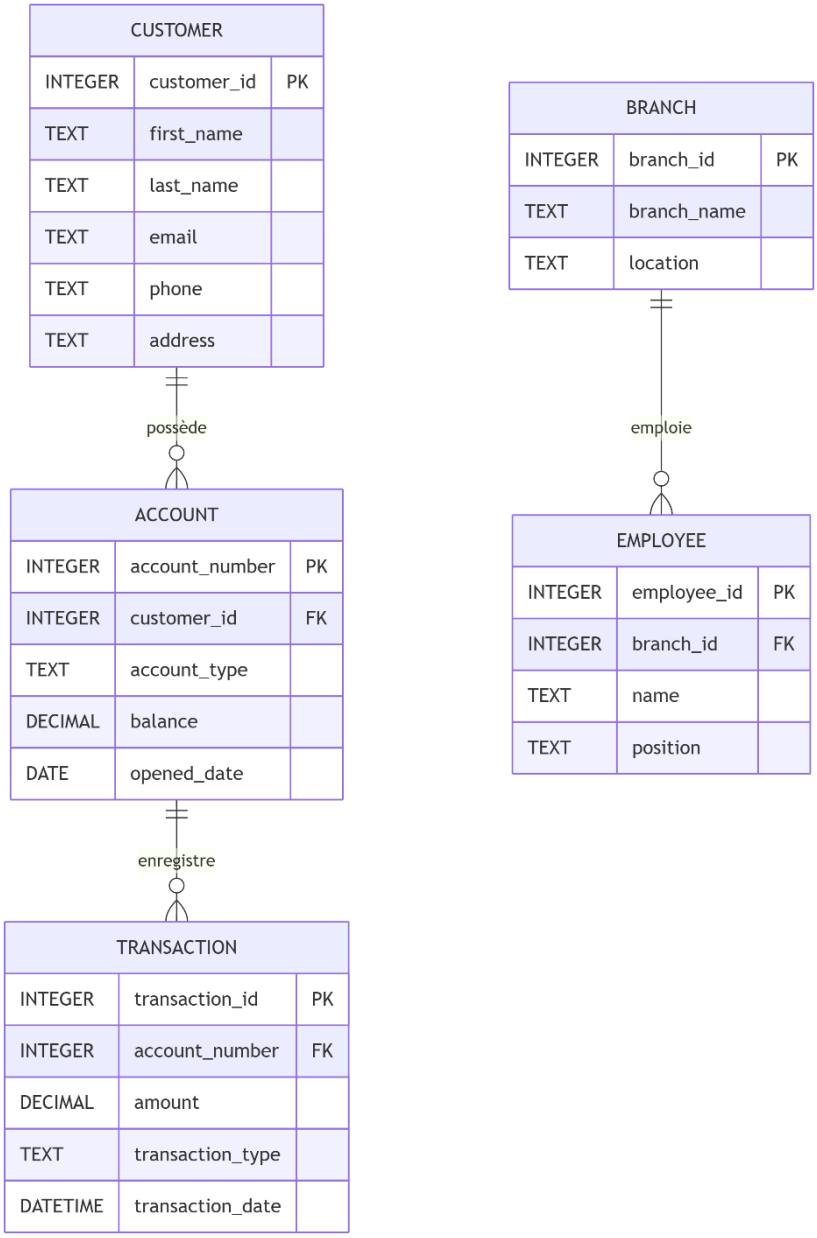
- La gestion des clients
- Les opérations financières
- Le suivi des succursales

Le système couvre d'abord la gestion des clients, chaque client est enregistré avec ses informations personnelles (nom, mail, adresse) et peut bénéficier de plusieurs comptes en même temps de différentes natures. Ensuite la gestion des transactions, consiste au fait que le système enregistre chaque opération (dépôt, retrait, virement) avec sa date/heure, et la lie à un compte en particulier. Pour éviter les bugs, des règles bloquent

les incohérences, comme le fait d'interdire un retrait si le solde est insuffisant. Enfin la gestion des agences, on implémente la base avec les données de chaque agence à savoir adresse et équipe. Ce qui permets à la fois d'organiser au mieux et de pouvoir comparer les résultats entre régions.

*Pour illustrer concrètement la structure logique décrite précédemment, un diagramme entité relation est présenté ci-dessous. Il formalise les principales entités du système, leurs attributs et les relations qui les unissent, en cohérence avec les trois domaines fonctionnels présentés. Ce schéma constitue la base de la modélisation conceptuelle et prépare l'implémentation physique de la base de données.*

3. Diagramme entité-relation



Il est important d'éclaircir les relations entre les entités, un client (CUSTOMER) peut posséder zéro ou plusieurs comptes (ACCOUNT), mais chaque compte est obligatoirement lié à un seul client. Cette décision reflète une simplification volontaire, excluant les comptes joints pour réduire la complexité initiale. Chaque transaction (TRANSACTION) est associée à un compte unique, assurant une traçabilité complète. L'absence de lien direct avec les clients évite les redondances (par exemple : le client peut être déduit via le compte). Les employés (EMPLOYEE) sont affectés à une seule succursale (BRANCH), permettant une gestion hiérarchique claire.

4. Optimisations de conceptions

La structure relationnelle suit une normalisation jusqu'à la troisième forme normale (3FN), éliminant les redondances, on peut retrouver un stockage unique des adresses clients par exemple. Les dépendances fonctionnelles sont isolées, il existe une liaison directe entre type de compte et identifiant de compte). Des contraintes d'intégrités on été mise en place afin de garantir une cohérence : validation des formats (emails via LIKE '%@'), restrictions sur les soldes (positifs pour les comptes courants, négatifs pour les prêts), et blocage des transactions nulles (amount != 0). Quant à l'intégrité des références elles est assurées par des clés étrangères avec suppression en cascade (ON DELETE CASCADE), préservant les liens entre clients, comptes et transactions.

Une indexation ciblée accélère les requêtes fréquentes : recherche de clients par email (Customer.email), consultation des historiques transactionnels par date (Transaction.transaction_date), ou analyse géographique des agences (Branch.location). Par ailleurs, l'utilisation de types de données spécialisés (exemple : DECIMAL(10,2) pour les montants, DATETIME pour l'horodatage) et de valeurs par défaut intelligentes (CURRENT_TIMESTAMP) réduit les erreurs manuelles. Les opérations critiques (exemple : virements) s'appuient sur des transactions ACID (BEGIN; COMMIT;), assurant atomicité et cohérence même en cas d'interruption.

La base de données pourrait par la suite faire l'objet de petites pistes d'évolutions telles que la normalisation des adresses via une table dédiée, le chiffrement des données sensibles (exemple : extension SQLCipher), ou l'archivage partitionné des transactions par année. Des colonnes d'audit (last_modified DATETIME) pourraient également être ajoutées pour tracer les modifications historiques.

5. Limitations et perspectives

Le modèle actuel, bien que fonctionnel, comporte des limitations intentionnelles pour nous faciliter la tâche, ce qui ouvre des perspectives d'évolution comme il a été introduit brièvement ci-dessus. Premièrement, la gestion statique des taux d'intérêt sur les comptes d'épargne et de prêt pourrait être dynamisée via une table annexe (*InterestRate*) associant taux, dates d'application et comptes concernés. Deuxièmement, l'absence de comptes joints, liée à la relation client-compte en 1:N, nécessiterait une table

d'association (*AccountOwnership*) pour autoriser une propriété partagée. Troisièmement, la sécurité des données sensibles (soldes, emails) reste perfectible par l'intégration de chiffrement natif (au niveau de la base ou de l'application). Enfin, si l'immuabilité des transactions respecte les normes bancaires, une colonne *cancellation_reason* permettrait de tracer les annulations sans altérer l'historique.

*Si les limitations identifiées appellent des évolutions futures, le modèle actuel n'en repose pas moins sur des fondamentaux solides, conciliant une structuration rigoureuse des données et une flexibilité d'exploitation adaptée aux besoins métier*.

6. Equilibre création/manipulation

Cet équilibre conceptuel se manifeste particulièrement dans :

- **La structuration robuste** : Un schéma normalisé et contraint assure la qualité des données lors de la création (ex : ouverture de compte).
- **La flexibilité d'interrogation** : Les index et relations bien pensées permettent des requêtes complexes (ex : solde total d'un client sur tous ses comptes) sans jointures excessives.

Exemple concret : la table Transactions est optimisée pour l'insertion rapide (colonne transaction_date par défaut CURRENT_TIMESTAMP), tout en permettant des analyses temporelles via son index dédié.