Donnez les réponses sur le sujet que vous joindrez à votre copie.

1 Détection de fraudes bancaires : contraintes et requêtes

Dans tous les exercices suivants, on fixe une base de données utilisée pour recouper des données bancaires et des données concernant certains propriétaires des comptes bancaires. La base de données respecte le schéma ci-dessous, où les clefs primaires sont soulignées et les clefs étrangères listées à la suite.

```
person(idP, nameP, age, cityP, countryP)
account(idA, idP, isBlocked)
transfer(idT, idA1, idA2, date, amount)
yatchClub(idY, nameY, cityY, countryY)
membership(idP, idY)
```

Clefs étrangères : `idP` dans `account` et dans `membership` font référence à `idP` dans `person` ; `idY` dans `membership` fait référence à `idY` dans `yatchClub`.

Les données concernant les propriétaires des comptes bancaires sont stockées dans la table `person`. Les transferts ont lieu d'un compte `idA1` vers un compte `idA2`. `IsBlocked` est un Booléen utilisé pour signifier que le compte a été utilisé à des fins de fraude fiscale et est donc bloqué.

Exercice 1 : schéma relationnel

Pour chacune des questions suivantes, écrivez `interdit` si le schéma ne permet pas les situations décrites. Sinon, expliquez quels choix faire (contraintes, types de données, etc.) à la création des tables pour que le schéma ne permette pas ces situations.

La syntaxe SQL précise n'est pas obligatoire : indiquez seulement la nature du choix, et les tables et attributs concernés.

1) Le compte source et le compte cible d'un transfert ne sont pas renseignés.

Solution: Autorisé. Ajouter une contrainte de non nullité sur chacun des deux attributs.

2) L'identifiant d'une personne apparait dans la table `membership` sans que l'identifiant du yatch club auquel elle appartient ne soit renseigné.

Solution: Interdit : aucun des attributs d'une clef primaire ne peut être nul.

3) Le compte source et le compte cible d'un transfert sont identiques.

Solution: Autorisé. Ajouter une contrainte vérification afin d'imposer l'inégalité (mot clef `CHECK`).

4) Une personne titulaire d'un compte bancaire peut être retirée de la table `person`.

Solution: Interdit si rien n'a été prévu en amont lors de la création de la contrainte de clef étrangère. Peut être autorisé avec `ON DELETE CASCADE`.

5) Si on change l'identifiant d'un yatch club dans la table `yatchClub`, cette modification est automatiquement répercutée dans la table `membership`.

Solution: Idem, ce type de comportement n'est pas prévu par défaut mais peut être implémenté lors de la création de la contrainte de clef étrangère en spécifiant `ON UPDATE CASCADE`.

Exercice 3 : requêtes SQL

Proposez des requêtes SQL permettant d'extraire les informations demandées.

1) Les noms des yatch clubs dont sont membres des propriétaires de comptes bloqués (tableau résultat : `nameY`).

Solution:

```
SELECT nameY
FROM yatchClub Y, membership M, account A
WHERE Y.idY=M.idY AND A.idP=M.idP
AND isBlocked;
```

2) La liste des villes dans lesquelles ne sont pas domiciliées des personnes possédant plus de 10 comptes différents, sans doublon (tableau résultat : `city`).

Solution:

```
SELECT DISTINCT P.cityP as city
FROM person P
WHERE NOT EXISTS (SELECT J.idP FROM (person NATURAL JOIN account) as J
WHERE J.cityP=P.city
GROUP BY J.idP HAVING COUNT(J.idA)>10);
```

Ou :

```
WITH big AS (SELECT idP FROM account
GROUP BY idP HAVING COUNT(idA)>10)
SELECT DISTINCT cityP as city FROM Personne WHERE cityP NOT IN
(SELECT cityP FROM big natural join account);
```

J'ai vu dans quelques rares copies une union du type suivant :

```
SELECT DISTINCT P.cityP as city
FROM person P
WHERE NOT EXISTS (SELECT J.idP FROM (person NATURAL JOIN account) as J
WHERE J.cityP=P.city
GROUP BY J.idP HAVING COUNT(J.idA)>10)
UNION
SELECT DISTINCT Y.cityY as city
FROM yatchclub Y
WHERE NOT EXISTS (SELECT J.idP FROM (person NATURAL JOIN account) as J
WHERE J.cityP=Y.city
GROUP BY J.idP HAVING COUNT(J.idA)>10);
```

C'est très bien, mais je ne vous en demandais pas tant. On reviendra dans la dernière partie du corrigé sur le fait que les différentes villes ne sont pas regroupées dans une table unique, ce qui procède en fait d'un défaut initial de modélisation du schéma.
Attention, il ne fallait pas confondre cette requête avec l'ensemble des villes dans lesquelles sont domiciliées des personnes ne possédant pas plus de 10 comptes différents :

```
SELECT DISTINCT cityP as city
FROM person
WHERE idP NOT IN (SELECT idP FROM person NATURAL JOIN account
GROUP BY idP HAVING COUNT(idA)>10);
```

Il a pu m'arriver de compter quand même compter les points par étourderie...

3) Les personnes qui appartiennent à tous les yacht club (tableau résultat : (`nameP`).

Solution:

```
SELECT nameP
FROM person NATURAL JOIN membership
GROUP BY idP HAVING COUNT(DISTINCT idY)=SELECT COUNT(idY) FROM yatchClub;
```

Cette version là est correcte également, bien que moins efficace sur de grosses données (chaque niveau d'imbrication de sous requête nécessite une coûteuse opération de jointure) :

```
SELECT nameP
FROM person P WHERE NOT EXISTS (SELECT * FROM yatchClub
WHERE idY NOT IN (SELECT idY FROM membership M WHERE M.idP=P.idP);
```

4) La liste des membres de yacht club qui ont réalisé des transfert de plus de 100000 euros vers le compte d'un autre membre du même yatch club (tableau résultat : `idP`).

Solution:

```
SELECT A1.idP
FROM account A1, account A2, transfert T, membership M1, membership M2
WHERE T.amount >1000000 and A1.idP=M1.idP AND A2.idP=M2.idP
AND A1.idA=T.idA1  AND A2.idA=T.idA2  AND M1.idY=M2.idY AND A1.idP<>A2.idP;
```

5) Le yatch club qui a le plus de membres titulaires de comptes bloqués (tableau résultat : (`idY`).

Solution:

```
WITH badguys AS (SELECT idY, idP
                 FROM account NATURAL JOIN membership
                 WHERE isblocked)
SELECT idY FROM badguys
GROUP BY idY
HAVING COUNT(DISTINCT idP)>=ALL (SELECT COUNT(DISTINCT idP)
                                 FROM badguys
                               GROUP BY idY);
```

ou (avec le max) :

```
WITH badguys AS (SELECT idY, COUNT(DISTINCT idP) as nb
                 FROM account NATURAL JOIN membership
                 WHERE isblocked
                 GROUP BY idY)
SELECT idY FROM badguys
WHERE nb = (SELECT MAX(nb) FROM badguys);
```

Exercice 4 : requêtes récursives avec postgres

1) Le compte d'idA 666 est bloqué. Il y a eu en particulier depuis ce compte un transfert particulièrement frauduleux de plus de 100000 euros qui interroge les enquêteurs. Cette somme semble avoir voyagé de compte à compte et on souhaite retracer son parcours. Proposez une requête SQL retournant tous les comptes connectés au compte 666 par une chaîne temporelle de transferts $t_1, t_2, ..., t_{n-1}$ (i.e., si $i < j$, t_i doit avoir eu lieu avant t_j). Evidemment on ne s'intéresse qu'aux transferts *significatifs* et chaque transfert dans la chaîne doit avoir fait transiter une somme importante, disons d'au moins 10000 euros. Vous retournerez seulement les identifiants de compte.

Solution:

```
WITH RECURSIVE chain (idA, date) AS (
SELECT idA2, date
FROM transfer
WHERE idA1=666 AND amount >= 100 000
UNION
SELECT T.idA2, T.date
FROM chain C, transfer T
WHERE C.idA=T.idA1 AND C.date < T.date AND T.amount >= 10 000
)
SELECT * FROM chain ;
```

On remarque ici que pour qu'il s'agisse bien d'une chaîne temporelle, il est préférable que l'attribut `date` ait été typé en amont de manière précise, au moyen par exemple d'un type `timestamp`.

2) Expliquez comment modifier votre requête de façon à identifier les *cercles de fraude*. On entend par cercle de fraude une séquence de transferts faisant transiter une somme importante de comptes en comptes, avec finalement retour à l'expéditeur.

Solution: (Question bonus) Une solution possible consiste à enregistrer le chemin sous forme de chaîne de caractère en utilisant l'opérateur de concaténation :

```
WITH RECURSIVE chain (idA, date, path) AS (
SELECT idA2, date, '666'::varchar  || '-' ::varchar || idA2::varchar
FROM transfer
WHERE idA1=666 AND amount >= 10 000
UNION
SELECT T.idA2, T.date, C.path || '-' ::varchar || T.idA2::varchar
FROM chain C, transfer T
WHERE C.idA=T.idA1 AND C.date < T.date AND T.amount >= 10 000
)
SELECT * FROM chain  WHERE idA=666;
```

Notez qu'il n'y a pas de problème de résultats potentiellement infinis ici, puisque la séquence temporelle doit être strictement croissante. Le chemin aurait également pu être enregistré dans un tableau (cf documentation de postgres). Ce type est utilisable dans les requêtes, mais son utilisation est déconseillée en tant que type d'attribut au niveau du schéma logique (en particulier parce que les recherches associées sont mal optimisées).

Exercice 5 : requêtes en algèbre relationnelle

Proposez des requêtes d'algèbre relationnelle permettant d'extraire les informations demandées.

1) Les habitants de Monaco (`countryP`) n'appartenant à aucun yatch club (tableau résultat : `nameP`).

Solution:

$$\pi_{nameP}(\sigma_{cityP='Monaco'}(person)) - \pi_{nameP}(person \bowtie membership)$$

2) Les transferts de capitaux en provenance d'un membre de yacht club (tableau résultat : `idT`).

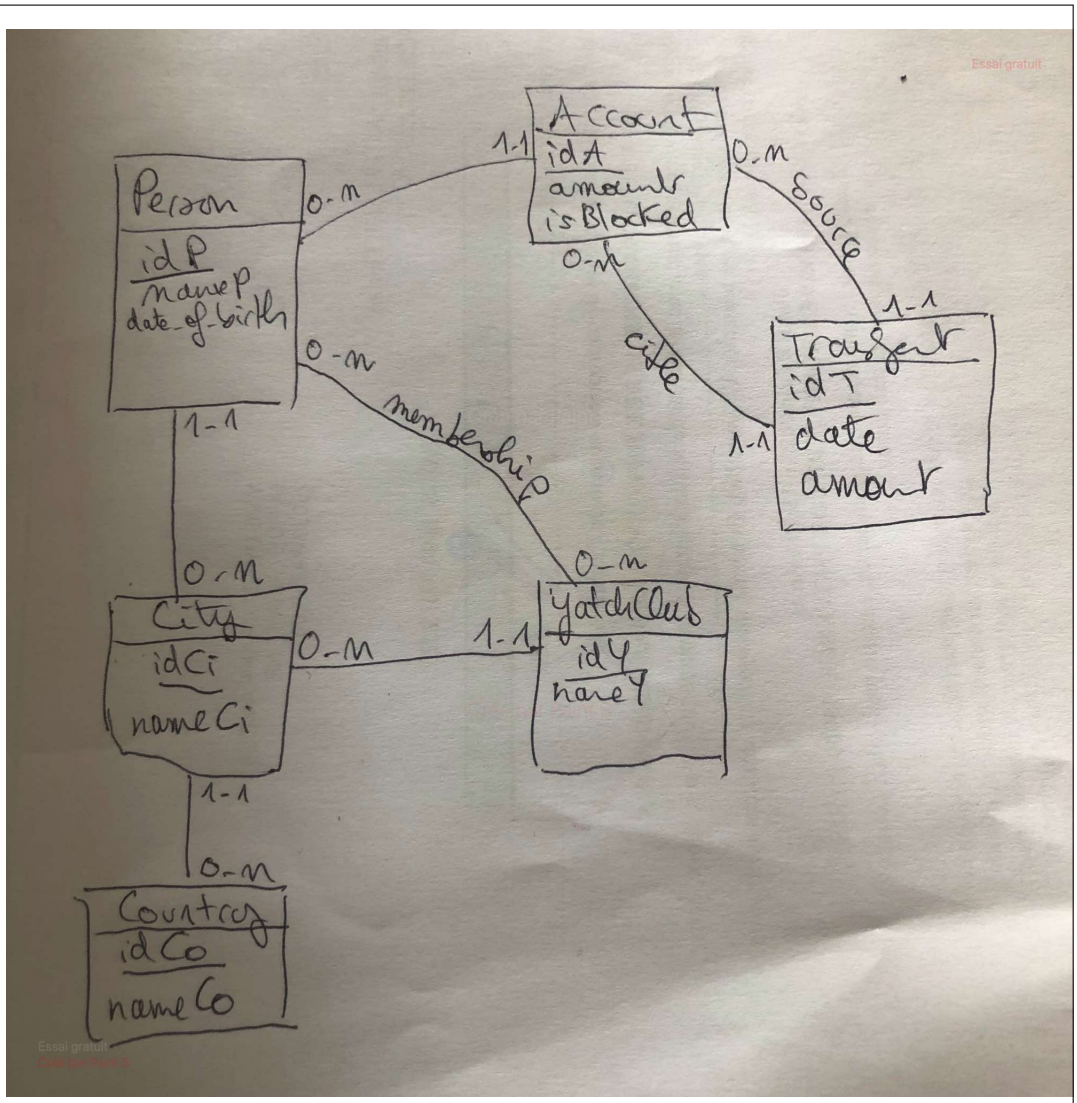
Solution:

$$\pi_{idT}((yatchClub \bowtie account) \bowtie_{idA=idA1} transfer)$$

2 Modélisation : reverse engineering, enrichissement et amélioration du schéma

1. Proposez une modélisation E/R correspondant au schéma relationnel décrit précédemment. Certains points sont problématiques (facteurs d'incohérence ou de redondance), améliorez-les. On précisera les entités, les identifiants, les associations, on indiquera pour chaque association les cardinalités. Listez les contraintes externes avec soin et expliquez comment les implémenter lorsque ceci est possible avec les notions vues en cours.

Solution:

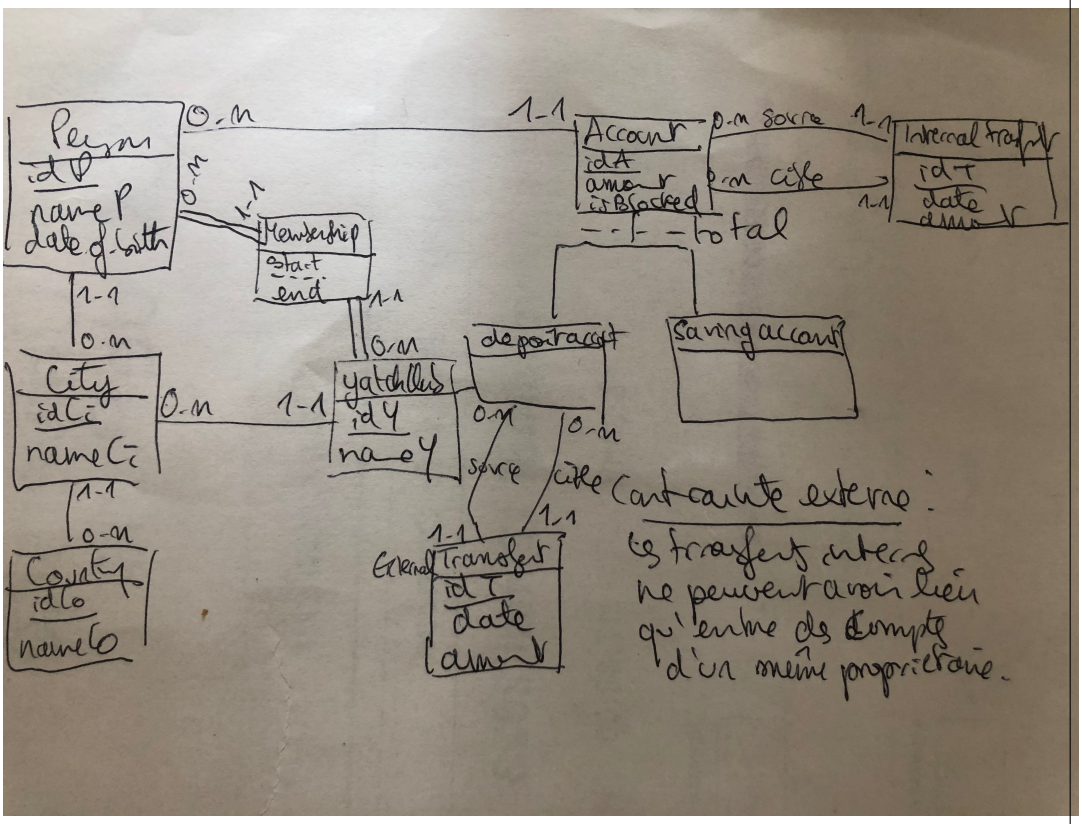


Utiliser une association plutôt qu'une entité pour représenter les transferts est une erreur : ceux-ci seront alors définis comme un sous-ensemble du produit cartésien Account × Account, ce qui implique qu'un unique transfert entre une même paire de comptes pourra être stocké (aucun virement récurrent ne sera donc possible : impossible par exemple de payer de façon mensuelle son fournisseur d'énergie à moins de créer un nouveau compte bancaire à chaque fois...). Notez qu'on ne représente pas directement l'attribut calculable `age`. Les attributs `country` et `city` étant présents dans deux tables différentes, il est par ailleurs préférable d'en faire des entités. Enfin, nous nous sommes permis d'ajouter un attribut `amount` dans l'entité `account`, car il est peu probable qu'un tel schéma soit utile sans cette information.
(Les contraintes externes seront listées bientôt.)

2. On décide de raffiner notre modélisation. Désormais il sera possible de suivre l'historique d'appartenance des personnes aux différents yatch clubs (date de début et date de fin). La notion de compte bancaire sera également modélisée de façon plus réaliste : seuls les "comptes de dépôt" (ou compte courant) pourront être impliqués dans des transferts impliquant les comptes de personnes différentes. Tout transfert de capital depuis un compte d'épargne ne pourra être réalisé que vers un compte (courant ou d'épargne) du même propriétaire.

   Modifiez le schéma E/R que vous avez proposé afin d'accommoder ces nouveaux besoins. N'oubliez pas de lister les contraintes externes.

Solution:



Notez ici l'usage d'une entité faible pour représenter `membership`. Ajouter simplement des attributs `start` et `end` pour les dates de début et de fin ne suffisait pas, car dans ce cas `membership` $\subseteq$ `person` $\times$ `yatchclub` et une même personne ne pourrait donc pas avoir appartenu au même yatchclub sur des périodes différentes. Parmi les contraintes externes additionnelles on peut noter en particulier que si une personne a appartenu au même yatchclub sur des périodes différentes, alors ces périodes devraient être disjointes. (Reste des contraintes externes à venir.)

Remarque : le trait reliant sur le dessin `deposit account` et `yatchclub` n'est pas volontaire et uniquement imputable à mes piètres compétences de dessinatrice.

3. Proposez une traduction de votre diagramme dans le modèle relationnel. Précisez les clefs primaires et les contraintes référentielles (en particulier les clefs étrangères). Si certaines contraintes du modèle n'ont pas pu être implementées, précisez-le.

Solution: Bien entendu, il ne fallait pas oublier l'étape de restructuration de la spécialisation du diagramme (détails à venir).