DS d'Informatique n° 2 : corrigé

PTSI Lycée Eiffel

26 mars 2021

Exercice 1

1. Beaucoup ici n'ont manifestement pas totalement compris le concept de clé primaire, surtout quand la clé primaire est un couple. Ici, si on choisissait le couple (npatient,nemploye) comme clé primaire, cela signifierait que le couple constitué des deux entiers doit être unique à chaque fois. Il serait donc tout à fait possible d'avoir un patient relié à plusieurs employés, et un employé relié à plusieurs patients, il ne faut surtout pas donner ça comme raison. Ce qui n'est pas possible, c'est d'avoir deux analyses différentes avec le même patient **et** le même employé. Or, cela aussi, ça peut très bien se produire ! Un même patient reviendra régulièrement faire de nouvelles analyses, et parfois avec un même employé, il faut donc autoriser la possibilité d'avoir un même couple (npatient,nemploye) pour plusieurs analyses différentes. On peut aussi considérer qu'il est préférable de séparer les différentes analyses effectuées par un patient lors d'une même prise de sang.
2. Certains qui ont trop lu mes corrigés des années précédentes utilisent INTEGER pour désigner la variable entière, mais c'est bien INT en SQL.
   CREATE TABLE Employes (nemploye INT PRIMARY KEY,
   nom VARCHAR(20),
   prenom VARCHAR(20),
   horaire VARCHAR(8) CHECK horaires IN ('matin','am','week-end')
3. (a) SELECT nom, prenom FROM Employes WHERE horaires='week-end'
   (b) SELECT COUNT(analyses) FROM Analyses WHERE date='2012-28-04'
   Beaucoup ici n'ont pas utilisé le format standard des dates en SQL (ce qui n'est pas bien grave).
   (c) SELECT nom FROM Patients WHERE prenom LIKE 'Jean-%'
   Attention, il faut absolument un LIKE et pas un simple = dans ce genre de requête. Quelques copies également où le principe du % qui remplace n'importe quelle chaîne de caractères n'a pas été compris.
   (d) SELECT nemploye FROM Employe JOIN Analyses ON Employes.nemploye=Analyses.nemploye JOIN Patients ON Analyses.npatient=Patients.npatient WHERE Patients.nom='Durand' AND type='Test PCR'
   Un certain nombre d'élèves n'utilisent pas de JOIN et font à la place une recherche dans plusieurs tables en les reliant dans le WHERE. Ils ont du réviser avec mes corrigés des années précédentes, ça fonctionne, mais le jury insiste chaque année pour voir des JOIN, donc forcez-vous à apprendre cette syntaxe (qui n'est pas plus compliquée). Par ailleurs, il faut faire attention aux attributs potentiellement ambigus : si on ne précise pas que c'est Patients.nom qui doit être Durand, comme il y a un autre attribut nom dans la table employe, la requête ne peut pas fonctionner. Certains ont aussi tenté une autre approche pour trouver le nom, en écrivant quelque chose du genre WHERE npatient=(SELECT npatient FROM Patients WHERE nom='Durand'). Pas bête, mais en fait, ça risque fort

de ne pas marcher. Pour le comprendre, il faut visualiser le fonctionnement de la requête SQL : le SELECT dans la parenthèse va afficher un tableau des numéros de patient de **tous** les patients qui s'appellent Durand. S'il n'y en a qu'un seul, pas de souci, mais s'il y en a plusieurs, demander à ce que ce tableau soit égal à un seul npatient n'a plus aucun sens.

(e) SELECT nanalyse FROM Analyses JOIN Patients ON Analyses.npatient=Patients.npatient JOIN Employe ON Employe.nemploye=Analyses.nemploye WHERE Patients.prenom=Employe.prenom

4. DELETE FROM Analyses WHERE date='2020-09-18' and type='Test PCR'

   Beaucoup trop de copies ici ont utilisé un ALTER TABLE, qui n'a pas du tout le même but : ALTER TABLE sert à modifier la structure d'une table (ajouter une colonne, en supprimer une) sans toucher aux données (sauf évidemment celles qui étaient dans une colonne supprimée par exemple). Ici, ce sont bien des **lignes** de données qu'on veut faire disparaitre. Notons également beaucoup de copies qui indiquent un (ou plusieurs) noms d'attributs derrière le DELETE. Ce n'est pas du tout utilise, la ligne va de toute façon être entièrement supprimée.

5. On va bien sûr créer une nouvelle table Medecins qui va contenir les informations concernant les mèdecins (typiquement un numéro d'identification, le nom du mèdecin, son adresse, etc). La commande de création de cette table n'était pas vraiment demandée. Ensuite, on doit créer le lien entre les mèdecins et les patients. L'association correspondante sera de cardinalité 1..n dans le sens Mèdecins ⇒ Patients (un mèdecin va certainement avoir plusieurs patients), mais 1..1 dans l'autre sens (le mèdecin traitant d'un patient est unique). On peut donc se contenter d'ajouter l'attribut nmedecin (numéro d'identification des mèdecins) à la table Patients (ce sera alors une clé secondaire de cette table, mais surtout pas le contraire comme beaucoup l'ont fait ! Si on veut être précis en SQL :

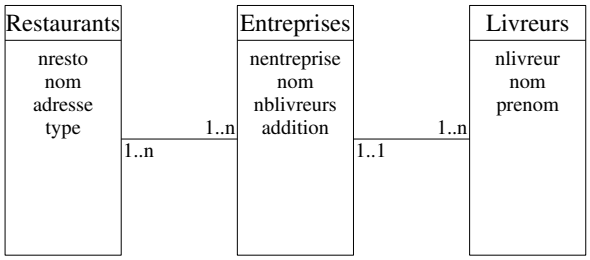
   ALTER TABLE Patients ADD nmedecin INT FOREIGN KEY References Medecins ON DELETE SET NULL

6. Peu de réponses vraiment très détaillées ici, mais le minimum consiste à :
   - créer une table **Maladies** recensant les maladies chroniques, avec nom de la maladie et éventuellement quelques attributs supplémentaires
   - relier les maladies et les patients en créant une nouvelle table **Malpat** ayant comme clé primaire un couple (nmaladie,npatient). Il faut en effet autoriser les patients à n'avoir aucune maladie, et surtout à en avoir plusieurs. Certains ont tenté de contourner ce problème en ajoutant dans la table Patients une colonne supplémentaire pour **chaque** maladie possible (avec un contenu booléen du type Oui/Non). C'est évidemment horriblement lours si on ne se limite pas à quelques maladies chroniques, et de toute façon déconseillé dans la mesure où les structures de bases de données sont faites pour ajouter des tables supplémentaires dans ce genre de cas.
   - il serait en fait bon d'ajouter en plus un lien entre les maladies et les analyses, donc une dernière table **Malana** qui aurait comme clé primaire un couple constitué de l'identifiant d'une maladie et d'un type d'analyses (et comme attribut principal la fréquence nécessaire pour le type d'analyse en question quand on a la maladie considérée), mais ça complique objectivement beaucoup les choses.

Exercice 2

1. Il suffit pour cette question de créer une entité pour chacun des trois éléments donnés par l'éoncé. Attention au niveau des associations : beaucoup d'entre vous ont créé trois associations, créant ainsi un « cycle » entre les trois entités. C'est le genre de choses qu'on ne fait pas dans ce modèle, la troisième association étant forcément inutile (on peut passer par la troisième entité pour relier les deux qui ne le sont pas directement). J'ai donc choici de relier Resturants et Entreprises avec une cardinalité 1..n dans chaque sens (comme l'impose

l'énoncé) et Entreprises et Livreurs avec une cardinalité 1..1 dans le sens Livreurs ⇒ Entreprises, puisqu'on avait précisé qu'un livreur ne travaillait que pour une seule entreprise. Les clés primaires ne sont pas précisées, mais il s'agit à chaque fois du premier attribut, créé spécialement pour les deux dernières entités :



2. Certains ici n'ont pas compris le principe du passage d'un modèle à l'autre, et ont simplement recopié la même chose que dans le schéma précédent (sous une autre forme, certes), ce qui n'a évidemment aucun intérêt. Rappelons donc que le modèle relationnel consiste à créer une liste de tableaux qui vont permettre de gérer à la fois les entités du modèle Entités/Associations, mais aussi les associations ! Ici, il faut donc absolument ajouter les choses suivantes :
   - pour le lien de cardinalité 1..1 1..n, on se contente d'ajouter une clé secondaire à la table Livreurs, en lui adjoignant un attribut nentreprise représentant l'entreprise dans laquelle le livreur travaille (qui est par hypothèse unique).
   - pour le lien complexe entre Resturants et Entreprises, il est indispensable de créer une nouvelle table qui aura une clé primaire double. En fait, je vais aller un peu plus loin, et créer (volontairement) une association entre Restaurants et Livreurs (et non pas Entreprises, ce qui ne change rien puisque les livreurs sont directement liés aux entreprises) avec une clé primaire indépendante (un même livreur peut livrer plusieurs fois de la nourriture du même restaurant), qui matérialisera une livraison donnée et permettra de faire beaucoup plus facilement la question 4 ensuite.

   Dans tous les cas, on doit avoir **quatre** tables dans notre bases de données, dans mon cas (en gras les ajouts par rapport au modèle précédent) :

   - Restaurants (nresto, nom, adresse, type)
   - Entreprises (nentreprise, nom, nblivreurs, addition)
   - Livreurs (nlivreur, nom, prenom, **nentreprise**)
   - **Livraison (nlivraison, nresto, nlivreur)**
3. Pas grand chose à signaler ici, comme toujours le langage de l'algèbre relationnelle est très lourd. Attention quand même à l'ambiguïté de certains attributs, notamment ceux qui s'appellent 'Nom' :
   (a) $\Pi_{\text{nom}}(\sigma_{\text{type='cuisine italienne'}}(\text{Restaurants}))$
   (b) $\Pi_{\text{Restaurant.nom}}(\text{Restaurants} \bowtie_{\text{Restaurants.nresto=Livraison.nresto}} \text{Livraison} \bowtie_{\text{Livraison.nlivreur=Livreurs.nlivreur}} \text{Livreurs} \bowtie_{\text{Livreurs.nentreprise=Entreprises.nentreprise}} (\sigma_{\text{nom='Deliveroo'}}(\text{Entreprises})))$
   (c) $\Pi_{\text{Livreurs.nom, prenom}}(\text{Livreurs} \bowtie_{\text{Livreurs.nlivreur=Livraison.nlivreur}} \text{Livraison} \bowtie_{\text{Livraison.nresto=Restaurants.nresto}} (\sigma_{\text{nom='Sushi delight'}}(\text{Restaurants})))$
4. Puisqu'on a eu la bonne idée de créer une table Livraison à la question 2, il suffit maintenant de la compléter pour prendre en compte le client concerné et les données supplémentaires. On aura désormais Livraison(nlivraison, nlivreur, nresto, nclient, date, montant) par exemple.
5. (a) Avec notre construction, on doit passer par les tables Livraison et Livreurs, ce qui alourdit un peu la commande :

SELECT adresse FROM Restaurants JOIN Livraison ON Restaurants.nresto=Livraison.nresto JOIN Livreurs ON Livraison.nlivreur=Livreurs.nlivreur JOIN Entreprises ON Livreurs.nentreprise=Entreprises.nentreprise WHERE Entreprises.nom='Just Eat'

(b) C'est une jointure un peu particulière ici : on n'a aucun élément en commun entre Livreurs et Clients a priori, mais il est totalement inutile de passer par l'intermédiaire des livraisons, tout ce qu'on veut c'est regrouper les deux tables en complémentant les lignes qui ont même nom et même prénom. Attention, on ne peut mettre qu'une seule condition derrière le JOIN, donc on aura quand même besoin d'un WHERE :

SELECT Livreurs.nom, Livreurs.prenom FROM Livreurs JOIN Clients ON Livreurs.nom=Clients.nom WHERE Livreurs.prenom=Clients.prenom

(c) Commande très lourde avec ce que je vous ai appris, puisqu'il faut bien sûr sélectionner le nom de l'entreprise, mais aussi le minimum du pourcentage, uniquement sur les entreprises liées à un restaurant de cuisine libanaise, ce qui force à écrire deux fois des conditions un peu lourdes.

SELECT Entreprises.nom FROM Entreprises JOIN Livreurs ON Entreprises.nentreprise=Livreurs.nentreprise JOIN Livraisons ON Livraisons.nlivreur=Livreurs.nlivreur JOIN Restaurants ON Restaurants.nresto=Livraison.nresto WHERE type='cuisine libanaise' AND addition=(SELECT MIN(addition) FROM Entreprises JOIN Livreurs ON Entreprises.nentreprise=Livreurs.nentreprise JOIN Livraisons ON Livraisons.nlivreur=Livreurs.nlivreur JOIN Restaurants ON Restaurants.nresto=Livraison.nresto WHERE type='cuisine libanaise')