Examen SD-202 bases de données, correction

Mardi 16 Juin 2020, 9h-12h

Documents et Internet autorisés.
Site du cours : `https://clarus.github.io/telecom-database-course/`

1 Arbres B

1. Les arbres B permettent de créer des indexes sur certains champs d'une table pour pouvoir rapidement accéder aux données. Leur structure est optimisée pour un stockage sur disque ou SSD où la mémoire se manipule par blocks, en allouant un grand nombre de valeurs dans chaque nœud. Les opération suivantes :
   — recherche
   — insertion
   — suppression

   ont pour complexité moyenne et en pire cas $O(ln(n))$.
2. Pour réaliser une insertion, on cherche où inserser notre élément dans les feuilles. Dans ce cas elle a lieu dans la feuille de droite $(7, 8, 9)$. Comme cette feuille est de taille maximale, on la coupe en deux (7) et $(9, 10)$ en faisant remonter l'élément central 8 dans le nœud $(2, 4, 6)$. En appliquant une nouvelle fois la même procédure, on coupe le nœud $(2, 4, 6, 8)$ en deux nouveaux (2) et $(6, 8)$ et en conservant bien les enfants de chacun. La valeur (4) devient la nouvelle racine.

2 SQL

1. Une requête SELECT a la forme :

```
SELECT champs groupés ou expressions d'agrégation
FROM tables
WHERE conditions sur les champs des tables
GROUP BY champs des tables pour grouper
HAVING conditions sur les champs groupés ou agrégats
ORDER BY tri sur les champs groupés ou agrégats ;
```

On extrait des lignes en réalisant le produit issu d'un certain nombre de tables, puis on filtre avec des conditions dans le WHERE, on regroupe certaines lignes en utilisant certaines expressions d'agrégation telles que la somme, on filtre de nouveau avec HAVING, et enfin on tri les résultats avec ORDER BY.

2. Voici un exemple de requêtes imbriquées, pour trouver les acheteurs qui n'ont pas acheté de produits alimentaires :

```
SELECT * FROM acheteurs
WHERE NOT EXISTS (
  SELECT * FROM achats
  WHERE achats.id_acheteur = acheteurs.id AND
    achats.type = alimentaire
)
```

3. — sélection $\sigma_{\text{condition}}$: SELECT * FROM table WHERE condition
   — projection π_{champ} : SELECT champ FROM table
   — jointure sur une colonne $\bowtie_c$: SELECT * FROM table1, table2 WHERE table1.c = table2.c

4. L'algèbre relationnel permet d'avoir une théorique mathématique simple pour représenter les requêtes sur un ensemble de relations, ainsi que de donner une stratégie calculable pour interpréter une requête. Les requêtes SQL sont cependant plus simple à écrire et à maintenir, en tant que langage déclaratif.

5. On peut utiliser SELECT DISTINCT pour avoir une liste de résultats sans doublon. En utilisant une contrainte d'unicité sur certaines colonnes ou des clés, on peut forcer une table à n'avoir que des lignes différentes.

3 Formes normales

Un attribut est atomique si il ne contient qu'une seule valeur pour un tuple donné, et donc s'il ne regroupe pas un ensemble de plusieurs valeurs.

1. Une date n'est généralement pas considérée comme une donnée atomique pouvant être encore découpée, malgré la présence de plusieurs sous-champs (année, mois, jour). Une date représente un point dans le temps et certaines bases de données fournissent un type dédié pour les dates.

2. On peut représenter une liste en donnant un élément de la liste à la fois et en dupliquant les autres éléments de la ligne. Par exemple, pour les noms suivants associés à certaines listes de commandes :
   — (Albert, [23, 565, 83])
   — (Jean, [6, 8])

on obtient la table :

Nom	Id
Albert	23
Albert	565
Albert	83
Jean	6
Jean	8

3. En utilisant l'algorithme de décomposition en forme BCNF, en prenant comme pivot "id_client" on obtient les tables :
   — ("id_client", "nom", "prenom", "adresse")
   — ("id_client", "id_commande", "date", "id_produit", "quantite", "prix")

   La première table est déjà en forme BCNF. Pour la seconde table, en prenant comme pivot "id_commande" puis "id_produit" on obtient :
   — ("id_commande", "id_client", "date")
   — ("id_produit", "prix")
   — ("id_commande", "id_produit", "quantite")

   Toutes les tables obtenues sont alors en forme BCNF.

4. — Une relation qui n'est pas en deuxième forme normale :
   (A, B, C) avec $AB \to C$ et $A \to C$
   car C est déterminé par une partie stricte A de la clé AB.
   — Une relation qui n'est pas en troisième forme normale :
   (A, B, C) avec $A \to B$, $A \to C$ et $B \to C$
   car C n'est pas dans une clé mais peut être déterminé à partir de B qui n'est pas une super-clé.
   — Une relation qui n'est pas en BCNF :
   (A, B, C, D) avec $ABC \to D$ et $A \to B$
   car B, bien que faisant partie d'une clé, peut être déterminé à partir de l'élément A qui n'est pas une super-clé.

5. Une version minimale :
   $$\{C \to AB,\ DE \to G,\ D \to CF,\ E \to H\}$$

6. Soit deux quadruplets (a, b_1, c, d_1) et (a, b_2, c, d_2) dans une relation R. Comme $A \to B$, alors $b_1 = b_2$. De $BC \to D$ appliqué aux deux quadruplets de la relation :
   — (a, b_1, c, d_1)
   — (a, b_1, c, d_2)

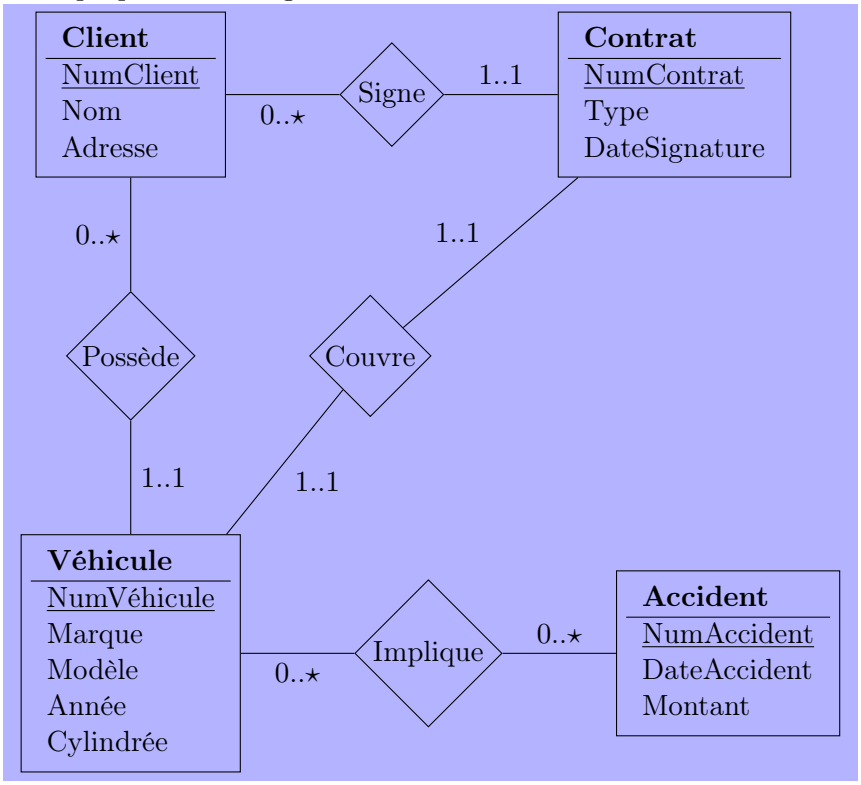
   on en déduit que $d_1 = d_2$ et donc on a la dépendance fonctionnelle $AC \to D$.

7. Si une relation au moins deux clés A et B différentes, alors il y a un chemin permettant d'aller de A à B, et un chemin permettant d'aller de B à A. Ainsi le graphe est cyclique.

Réciproquement, un graphe peut avoir un cycle sans avoir de clé, s'il n'est pas connexe par exemple.

4 Modèle entité-association

1. Nous proposons le diagramme entité-association suivant :



2. On peut ajouter un attribut « pourcentage d'implication » à l'association reliant accident et véhicule.

3. — la liste des accidents :

```
SELECT DISTINCT AccidentImpliqueVéhicule.NumAccident
FROM AccidentImpliqueVéhicule, Véhicule
WHERE
  AccidentImpliqueVéhicule.NumVéhicule = Véhicule.NumVéhicule AND
  Véhicule.Cylindrée > 6
```

   — le nombre de véhicules par client :

```
SELECT Véhicule.NumClient, COUNT(Véhicule.NumVéhicule)
FROM Véhicule
GROUP BY Véhicule.NumClient
```

   — la marque de véhicule ayant le plus d'accidents :

```
SELECT
  Véhicule.Marque,
  COUNT(AccidentImpliqueVéhicule.NumAccident) AS NbAccidents
FROM Véhicule, AccidentImpliqueVéhicule
WHERE AccidentImpliqueVéhicule.NumVéhicule = Véhicule.NumVéhicule
GROUP BY Véhicule.Marque
ORDER BY NbAccidents DESC
LIMIT 1
```

— les clients qui possèdent un véhicule de chaque marque :

```
SELECT * FROM Client
WHERE NOT EXISTS (
  SELECT * FROM Véhicule AS V1
  WHERE NOT EXISTS (
    SELECT * FROM Véhicule AS V2
    WHERE
      V1.Marque = V2.Marque AND
      V2.NumClient = Client.NumClient
  )
)
```