Cours et TP bases de données
ECG

Julien Reichert

2024/2025

Introduction

Ce document présente les notions utiles pour l'enseignement de bases de données en ECG. Il s'agit d'une réécriture adaptée de mon cours de tronc commun.

L'objectif de cet enseignement est essentiellement de donner des techniques pour manipuler dans les cas simples une base de données par l'utilisation du langage SQL. Cependant, des bases théoriques sont indispensables pour mettre en contexte les manipulations qui seront effectuées.

Apparues dans les années 1970, les bases de données sont de plus en plus présentes dans la gestion assistée par l'informatique du quotidien.

En ce qui concerne le monde commercial, le catalogue, les stocks et les clients d'un magasin sont l'exemple qui viendra le premier en tête. Dans l'optique des séances de TP associées, c'est une telle base de données qui sera utilisée.

Vocabulaire

Soit un objet caractérisé par un certain nombre d'*attributs*. On va considérer cet objet comme le n-uplet formé par les valeurs de ces attributs dans un ordre fixé pour chaque objet.

Une *relation* est un ensemble fini d'objets (appelés ici *valeurs* ou *enregistrements*). On précisera habituellement la structure de cette relation.

Le *domaine* d'un attribut est l'ensemble de ses valeurs possibles (disons un type).

Un *schéma relationnel* est la précision de la structure d'une relation, sous la forme de n-uplet formé des couples (attribut, domaine) dans l'ordre.

Afin de permettre une représentation efficace d'une relation, on utilisera simplement un tableau (au sens habituel du terme), d'où le nom de *table* que l'on verra dans la section sur les bases de données.

Les opérations sur les relations, ou *opérateurs relationnels*, sont essentiellement des recherches d'enregistrements ou se fondent dessus en effectuant un traitement sur les résultats.

Il s'agit essentiellement d'opérations ensemblistes : réunion, intersection, différence, produit cartésien.

Le produit cartésien a un intérêt particulier, car il permet, en filtrant les couples d'éléments issus de deux tables sur lesquelles ce produit cartésien a été fait, de faire le lien entre des attributs de chacune. Disposer de tables séparées mais sur lesquelles on fait un produit cartésien présente en particulier l'avantage de permettre de disposer d'un stockage des mêmes données de manière plus concentrée.

Notion de clé

Il est recommandé que les tables dans les bases de données soient munies de *clés*.

Il s'agit d'attributs (parfois de n-uplets d'attributs) dont aucune valeur du domaine ne peut apparaître deux fois dans la table.

Ceci permet aussi parfois d'optimiser les recherches quand la table est triée en fonction d'une clé (penser à la dichotomie).

Deux notions liées aux clés sont au programme : la *clé primaire* qui est simplement une des clés de la table pour laquelle les recherches sont justement optimisées, et la *clé étrangère*, qui n'existe pas forcément dans une table, et qui est simplement une valeur correspondant à une clé d'une autre table, afin de relier les deux tables concernées.

Le langage SQL

Interroger une base de données se fait par le moyen de *requêtes*, que nous écrirons dans le langage SQL. Ceci offre plus de puissance et de confort qu'une manipulation directe de tableur, ce qu'on aurait pu intuiter comme méthode de gestion de données.

L'essentiel des requêtes formulées seront des recherches d'enregistrement à partir de propriétés sur les attributs.

Les requêtes principales concernant les données se regroupent en quatre catégories : insertion, recherche, mise à jour et suppression.

Curieusement, la création de tables est également au programme alors qu'elle a toujours été absente en CPGE scientifiques. La syntaxe est :

```
CREATE TABLE ‘Nom_table‘ ( ‘nom_attribut‘ TYPE ,
‘autre_nom_attribut‘ TYPE , ... )
```

Attention, il s'agit d'accents graves (Alt Gr + 7) et non pas d'apostrophes. En pratique on peut ne pas en utiliser s'il n'y a pas d'espace dans les noms, espaces qui sont de toute manière à éviter.

La partie au programme semble s'arrêter après le nom de la table. Cette requête ne sera par ailleurs pas utilisée dans le TP, les tables étant prêtes.

L'insertion se contente d'ajouter une entrée dans la table, en renseignant tous les attributs nécessaires.

Les autres requêtes nécessitent de préciser à l'aide de conditions sur quel(le)s entrée(s) l'opération doit être effectuée. Ces conditions portent sur les attributs de la table : comparaisons, tests d'égalité, essentiellement.

Si des calculs sont impliqués, les opérations arithmétiques `+`, `-` et `*` sont disponibles (la division n'est pas au programme, on fera une multiplication pertinente si nécessaire).

Pour les comparaisons, les opérateurs sont `=` (test d'égalité), `<>` (test de différence), `<`, `<=`, `>` et `>=`.

Les opérations booléennes utilisent les mots-clés `AND`, `OR` et `NOT`. L'utilisation de capitales est conventionnelle mais non obligatoire, et ceci vaut pour tous les mots-clés.

La syntaxe SQL de ces requêtes sera présentée au fur et à mesure de l'avancement du TP et en fonction des besoins.

Le TP

Ce TP aura lieu sur la plate-forme d'administration de mon hébergeur. L'adresse est `http://phpmyadmin.online.net`, l'identifiant est `db86896` et le mot de passe est communiqué au cours de la séance. La navigation sur le site sera expliquée à l'oral.

On utilisera dans un premier temps une base de données simpliste pour gérer un commerce en ligne (bien entendu un vrai commerce en ligne ne pourrait pas se contenter des tables que nous utiliserons), et le remplissage de la base de données sera fait au fur et à mesure de la séance par des requêtes. Cette base de données comprend trois tables. **Depuis 2023, elle contient d'autres tables pour rassembler des bases de données d'enseignement faute de place. On ne se concentrera que sur celles qui sont détaillées ci-après.**

— `Clients`, avec comme attributs `id_client` (entier et clé primaire), `identite` (chaîne de caractères), `email` (chaîne de caractères), `adresse` (chaîne de caractères) et `telephone` (chaîne de caractères).
— `Produits`, avec comme attributs `id_produit` (entier et clé primaire), `nom_produit` (chaîne de caractères), `categorie` (chaîne de caractères), `stock` (entier) et `description` (texte potentiellement long).
— `Commandes`, avec comme attributs `id_commande` (entier et clé primaire), `id_client` (entier), `id_produit` (entier) et `quantite` (entier).

La structure est relativement intuitive.

Exercice 1 : Créer des clients, des produits et des commandes. Quel problème peut se poser au moment de la création d'une commande ? Comment peut-on imaginer le résoudre ? (Une réponse sera apportée par la suite.)

Pour ajouter une entrée à la base de données, la syntaxe est

```
INSERT INTO ‘Nom_table‘
VALUES ('valeur_attribut', 'valeur_autre_attribut', ...)
```

en renseignant tous les attributs dans l'ordre fourni au niveau du schéma de la table. **Attention aux apostrophes cette fois au niveau des valeurs (pour les entiers, elles ne sont pas nécessaires).**

En pratique (pas à savoir), pour notre base de données, les clés primaires n'ont pas à être renseignées car elles seront gérées automatiquement pour garantir que toutes les valeurs soient différentes, et surtout tenter une valeur existant déjà posera un problème.

La syntaxe adaptée est :

```
INSERT INTO ‘Nom_table‘(‘attribut1‘, ‘attribut2‘, ...)
VALUES ('valeur_attribut1', 'valeur_attribut2', ...)
```

en renseignant tous les autres attributs que la clé (ici tous sont nécessaires, et la notion de valeur par défaut ne sera pas abordée).

Exercice 2 : Une fois suffisamment de clients créés, afficher tous les clients.

La syntaxe pour récupérer le contenu d'une table est :

```
SELECT * FROM ‘Nom_table‘
```

Exercice 3 : Une fois suffisamment de produits créés, afficher tous les noms de produits.

Pour ne récupérer qu'un attribut de tous les éléments d'une table, la syntaxe s'adapte en :

```
SELECT ‘attribut‘ FROM ‘Nom_table‘
```

... et en pratique on peut récupérer autant d'attributs qu'on veut en séparant à chaque fois par des virgules.

Une version plus complète de la récupération d'un attribut est de préfixer par le nom de la table, et ceci est nécessaire dans le cadre de jointures si un attribut porte le même nom dans plusieurs tables :

```
SELECT ‘Nom_table‘.‘attribut‘ FROM ‘Nom_table‘
```

(Le préfixage est toujours autorisé, mais on se limite aux cas où c'est nécessaire pour alléger et gagner du temps.)

Exercice 4 : Afficher tous les produits dont le nombre d'exemplaires en stock est strictement inférieur à 5.

Pour filtrer le contenu d'une table en fonction d'une condition, on introduira cette condition par le mot-clé `WHERE` après la mention de la table où la recherche se produit.

Exercice 5 : Choisir un produit qu'on a créé et changer le nombre d'exemplaires en stock pour qu'il devienne `19`.

La syntaxe est la suivante :

```
UPDATE ‘Nom_table‘
SET ‘attribut‘ = ’valeur’
WHERE condition
```

Ici aussi on utilisera une condition introduite par `WHERE`, et le nombre de mises à jour est également quelconque, en séparant les mises à jour par des virgules.

Exercice 6 : Choisir une commande qu'on a créée et la supprimer.

La syntaxe est la suivante :

```
DELETE FROM ‘Nom_table‘
WHERE condition
```

Attention à ce que la condition caractérise exactement ce qui doit être supprimé. C'est d'ailleurs tout aussi important pour la mise à jour qui ne devrait pas impacter d'autres enregistrements. En pratique, c'est aussi pour cette raison que toutes les tables ont une clé.

Exercice 7 : Choisir un client qu'on a créé et afficher toutes les commandes qu'il a passées.

Pour faire un produit cartésien de deux tables, on fait suivre le mot-clé `FROM` non plus seulement du nom d'une table mais d'une jointure, dont la syntaxe est

```
‘Nom_table1‘ INNER JOIN ‘Nom_table2‘
```

... cependant ici tous les enregistrements d'une table vont être fusionnés avec tous les enregistrements de l'autre, créant des couples qui n'ont aucune pertinence, donc la plupart du temps les jointures s'accompagneront d'une condition d'égalité entre deux attributs.

Il est plus sensé que ces attributs soient la clé primaire d'une des tables et une clé étrangère de l'autre se référant à cette clé primaire. Ici les noms d'attribut ont mis en lumière les clés étrangères.

La syntaxe à ajouter après le passage avec `INNER JOIN` est :

```
ON ‘attribut1‘ = ‘attribut2‘
```

... mais il est équivalent (quoique moins apprécié) de mettre cette condition dans le `WHERE`, éventuellement conjoint par `AND` à la condition déjà existante.

Exercice 8 : Compléter la requête de l'exercice 7 pour avoir des renseignements sur les produits achetés.

Il s'agit d'enchaîner deux jointures pour couvrir les trois tables.

C'est tout pour la découverte des commandes exigibles !

À présent, deux exercices un peu plus compliqués pour terminer.

Exercice 9 : Écrire une requête qui affiche toutes les commandes de produits pour une quantité déjà strictement supérieure au stock.

Exercice 10 : Adapter la requête pour afficher seulement l'identité de chaque personne ayant passé une telle commande.