Chapitre II Base de données

Introduction: depuis, les années 1970 et l'avènement de l'informatique, la quantité de données numérique produite par notre société a augmenté de manière exponentielle (supermarché WalMart gère 10^{15} octets de données par heure, Facebook traite 3 milliards de "pouce bleu" par jour, etc...).Le VOLUME des données créées double tous les 2 ans. Toutes ces données sont stockées et manipulées dans des bases de données.

I) Qu'est-ce qu'une base de données ?

1) Exemples de données

- dossiers médicaux
- billetteries spectacles ou voyages
- comptabilité d'entreprise
- état des stocks dans une entreprise
- système d'informations schengen: renseignements judiciaires
- catalogue itis de toutes les espèces vivantes
- CIA
- librairie Amazon (250 millions d'ouvrages, films)
- forums, messageries
- antivirus
- data.gouv.fr
- gestion des bibliothèques

2) Différentes bases de données

Les données peuvent être organisées de différentes façons:

- base de données relationnelle (très structurée). Les données sont regroupées en tables appelées aussi relations. Les tables peuvent être connectées entre elles en partageant une colonne. (voir illustration)
- bases de données orientées "document","graphe" ou "clé:valeur" (moins structurées)

Leur stockage et leur manipulation sont réalisées par des logiciels appelés SGBD (systèmes de gestion de base de données)comme: mySQL, postreSQL, Oracle, SQLServer, DB2 d'IBM pour les relationnels et noSQL, couchDB ou mongoDB pour les non relationnels.

II) Les bases de données relationnelles

1) Organisation en tables: colonnes, lignes et vocabulaire

voir ilustration
Une base de données relationnelle organise les données en tables (1 table = 1 relation).
Exemple: base de données d'une bibliothèque organisée en 3 tables: Livres, Usagers et Emprunts.
Une ligne d'une table contient un **tuple** ou n-uplet comme le triplet:('La programmation en pratique', 'Alice', '01/04/20')
Une table ou relation est un ensemble de tuples.
Chaque tuple est unique
La table est constituée de colonnes. Une colonne correspond à un **attribut** ou champ. Par exemple, 'Titre' est un attribut de la table Emprunts.
Les valeurs des attributs doivent toutes êtres du même type: par exemple, tous les titres sont des chaines de caractères, tous les code-barres sont des entiers. Les valeurs d'un attribut appartiennent à un **domaine**. Le domaine représente l'ensemble des valeurs valides pour cet attribut
Par exemple, l'attribut retour est une date dont l'annéé ¡2020 et ¿date création bibliothèque !
Pour résumer la structure d'une table, on utilise un **schéma** qui indique le nom et le domaine de chaque attribut d'une table
Exemple: schéma de la table Livres:
Livrestitre String, auteur(s) String, éditeur String, année Int, isbn String
On parle aussi de schémas pour une base de données: c'est l'ensemble des schémas des tables de la base.
Pour éviter les doublons pour un attribut A d'une table T1, il est fortement recommandé de créer une autre table T2 avec cet attribut A et un attribut de type id (entier toujours différent). On remplace l'attribut A de T1 par id.

2) Clé primaire

Schéma global de la base de données cinema. C'est quoi une clé primaire ? id_film, id_nationalite, id_genre sont des clés primaires.

La clé primaire est un attribut particulier d'une table car, dans la colonne de cet attribut, les valeurs doivent être toutes différentes (aucun doublon). La clé primaire permet de s'assurer que tous les t-uplets de la table sont uniques (contrainte d'entité)et donc de pouvoir accéder à 1 enregistrement sans ambiguité (contrainte d'entité).

Dans la table Usagers du document, le code barre est une clé primaire puisque sa valeur est différente à chaque inscription (création d'une carte) à la bibliothèque.

Quels problèmes poserait le choix de l'attribut adresse email comme clé primaire dans la table Usagers ?

Dans la table Livres, quel pourrait être le choix de la clé primaire ? L'ISBN (International Standard Book number) paraît être satisfaisant mais...non car l'ISBN est unique pour un ouvrage d'un auteur mais il est identique pour 2 copies de cet ouvrage ! Il faudrait que la bibliothèque ajoute un code barre sur chaque exemplaire de tous les livres.id_livre est est une clé primaire valide.

3) Clé étrangère

Dans la base de données cinémas, pour la table film, id_nationalite, id_realisateur, ordre_affichage sont des clés étrangères. Cela permet de faire référence à une unique nationalité ou un genre unique.

Les clés primaires ne permettent pas seulement de distinguer les entités de manière unique.Elles permettent aussi de servir de référence dans une autre table.

La clé étrangère d'une table A est un attribut devant être une clé primaire pour une table B. Cela permet de faire référence dans un t-uplet de la table A à un unique t-uplet de la table B.

La table emprunts est naïve. Il vaut mieux la définir de cette façon:

	Attributs	types	domaines	
Emprunts	code_barre	chaine	x car	avec id_livre comme clé primaire
	id_livre	chaine	13 car	
	retour	date	date	

code_barre et id_livre sont des clés étrangères.Cela permet de n'emprunter que des livres qui existent dans la table par des personnes qui existent aussi dans celle-ci !
Cela oblige à ce qu'une personne rende tous ses livres avant de se désinscrire. Supprimmer d'abord l'usager aurait rompu la contrainte de référence: le code_barre dans usagers doit correspondre à un code_barre dans emprunts.
A noter que dans la table emprunts, id_livre est une clé primaire.
Une clé étrangère peut être une clé primaire.

4) Contraintes d'intégrité

Il faut respecter un certain nombre de contraintes pour que les données restent cohérentes. Les contraintes d'intégrités sont des règles que doivent respecter les données à tout instant.

- contrainte d'entité: chaque ligne doit être unique (au moins une donnée différente)
- contrainte de domaine: chaque donnée d'un attribut est d'un seul type et d'une longueur max définie
- contrainte de référence: avec une clé étrangère, on ne peut pas supprimer la table référencée sans conséquence.
- contrainte utilisateur: au cas par cas: un @ dans email

III) Langage SQL

Le SQL (Structured Query Language) est un langage qui permet de créer des tables, d'insérer des enregistrements, de les supprimer, de les sélectionner, de faire des calculs à partir des données.Une requête sera écrite en sql: c' est une demande adressée à une base de données.SQL est un langage déclaratif: il dit ce qu'il faut faire mais pas comment il faut le faire. Il est impératif aussi. Les ordres se suivent les uns après les autres.

1) Création d'une table en SQL

Exemple: CREATE TABLE eleve (id_eleve INTEGER NOT NULL PRIMARY KEY, nom char(45), prenom char(45), spe1 char(45), spe2 char(45), ddn date); Il faut donc spécifier le nom de la table, les attributs, les types (char ou varchar) et les contraintes (PK ou FK). Habitudes: le point-virgule, majuscules pour les mots clés et minuscules pour les attributs, pas d'espace dans les attributs donc _, nom de table au singulier.

2) Destruction d'une table en SQL

DROP TABLE eleve;

Cela supprime toutes les données. Impossible de créer une autre table avec le même nom. Impossible de supprimer une table qui sert de référence pour une clé étrangère d'une autre table (casse la contrainte de référence). IL faut d'abord supprimer les tables contenant les clés étrangères avant les tables contenant les clés primaires référencées.

3) Insertion dans une table en SQL

Insert into eleve values (...) ordre avec: Insert into eleve (....) values (...) Attention les contraintes d'intégrité sont rappelées à ce moment là !

IV) Requêtes SQL et mises à jour

Nous avons appris à créer des tables (CREATE....), à les détruire (DROP...) et à insérer des données (INSERT into...). Nous allons découvrir maintenant ce que l'on fait une fois la base construite: sélectionner et mettre à jour des données. Nous utiliserons la base de données d'une bibliothèque dont voici la structure:

1) Sélection des données

1).1 Principe

Pour retrouver certaines lignes, on utilise la commande SELECT Commençons par une requête simple: SELECT titre FROM livre WHERE annee >= 1990;
Partie SELECT titre: on affiche que les valeurs titre des lignes trouvées.
Parite FROM livre: table sur laquelle porte la requête.
Partie WHERE : critère de sélection. Le résultat d'une requête est une nouvelle table.

1).2 Clause WHERE

L'expression se trouvant dans la partie WHERE doit être un booléen.Elle peut être construite à partir d'opérateurs de comparaisons $>, <, =, ! =$ *ou* $<>$, d'opérateurs mathématiques ($+, -, /, *, \%$), d'opérateurs logiques (AND, OR et NOT) et d'opérateur de comparaison de textes comme LIKE.
Exemple: SELECT titre FROM livre WHERE annee ¿= 1970 AND annee ¡= 1990 AND editeur="Dargaud";

Like permet de sélectionner des titres de livre qui contiennent certaines chaines de caractères.
SELECT titre from livre WHERE titre LIKE '

% toute chaine de caractères

1).3 mot-clés DISTINCT et ORDER BY

Dans notre base de données biblio, demandons la liste des noms de tous les auteurs:
SELECT nom FROM auteur;
Cette requête nous renvoie 119 tuples. Elle n'est pas très lisible. Affichons-la par ordre alphabétique croissant:
SELECT nom FROM auteur ORDER BY nom;

ORDER BY permet de trier les tuples. Par défaut, ORDER BY trie par ordre croissant. Pour trier par ordre décroissant, il faut ajouter DESC (pour descendant).
SELECT nom FROM auteur ORDER BY nom DESC;

A noter ici que "Otomo" rejeté en fin d'alphabet (premier caractère spécial) et iMinds car commence par une minuscule.
Evidemment, ORDER BY trie aussi des nombres:
SELECT titre,annee FROM livre ORDER BY annee;**voir doc illustrations n°1**

Revenons à notre requête de sélection d'auteurs: SELECT nom FROM auteur ORDER BY nom;
On remarque qu'elle renvoie, pour certains auteurs, plusieurs fois le même nom. Par exemple, elle renvoie 3 fois "Goscinny". Il y a en effet 3 tuples possédant la valeur "Goscinny" pour l'attribut nom. Pour ne récupérer qu'une fois les tuples ayant un attribut de nom identique, il faut ajouter le mot DISTINCT:
SELECT DISTINCT nom FROM auteur ORDER BY nom; **voir doc illustrations n°2**

"DISTINCT a" dans une requête SQL permet de ne récupérer qu'une fois les tuples ayant des valeurs de champ a égales. Attention DISTINCT * renvoie tous les tuples si ils contiennent une clé primaire.S'ils n'en contiennent pas, DISTINCT * retournera les tuples qui ont au moins une valeur d'attribut différente.

voir doc illustrations n°3

1).4 Jointure

Imaginons que je veuille lire tous les livres de la bibliothèque de Isaac Asimov. Je serai tenté d'exécuter la requête:
SELECT titre FROM livre WHERE auteur="Asimov";
Cette requête me renvoie une erreur puisqu'il n'y pas de champ auteur dans la table livre. Par contre, dans la table livre, il y a l'attribut isbn et on retrouve cet isbn dans la table auteur_de. On peut donc joindre les 2 tables au niveau de l'attribut isbn. La syntaxe est la suivante:
SELECT titre,livre.isbn, auteur_de.a_id FROM livre JOIN auteur_de ON livre.isbn= auteur_de.isbn;
Cette fois la requête nous retourne une table contenant le titre, l'isbn et l'id de l'auteur. **voir doc illustrations n°4**

Etant données 2 tables A et B, la jointure consiste à créer toutes combinaisons de lignes de A et de B ayant un attribut de même valeur.La syntaxe de la jointure est du type JOIN A ON A.attribut = B.attribut. Le résultat d'une jointure est une table.

remarque: on utilise la notation table.attribut pour différencier les colonnes de même attribut

de 2 tables différentes.

Pour pouvoir accéder au titre d'un auteur, il faut faire une nouvelle jointure avec la table précédente et la table auteur sur l'attribut a_id:
SELECT titre,livre.isbn, auteur_de.a_id, auteur.nom FROM livre JOIN auteur_de ON livre.isbn= auteur_de.isbn JOIN auteur ON auteur.a_id = auteur_de.a_id ORDER BY nom; **voir doc illustrations n°5**

Il suffit maintenant de préciser l'auteur:
SELECT titre,livre.isbn, auteur_de.a_id, auteur.nom FROM livre JOIN auteur_de ON livre.isbn= auteur_de.isbn JOIN auteur ON auteur.a_id = auteur_de.a_id WHERE nom='Asimov' ORDER BY nom; **voir doc illustrations n°6**

2) Modification des données

2).1 Mise à jour des données avec UPDATE

Dans la base monde.db du tp SQL jointures, s'il recherche toutes les données sur la France avec la requête:
SELECT * from Pays WHERE Pays.Nom = 'France';

On obtient:

	Code	Nom	Continent	Region	Superficie	Population	EspeVie	NomLocal	TypeGouvernance	ChefEtat
1	FRA	France	Europe	Europe de l'Ouest	551500.0	59225700	78.8	France	République	Jacques Chirac

Une petite mise à jour de l'attribut chefEtat s'impose ! La requête nécessaire dans ce cas est:
UPDATE Pays SET ChefEtat ='Emmanuel Macron' WHERE Pays.Nom = 'France';

Pour vérifier qu'un changement a eu lieu, il faut relancer la requête précédente:
SELECT * from Pays WHERE Pays.Nom = 'France';

	Code	Nom	Continent	Region	Superficie	Population	EspeVie	NomLocal	TypeGouvernance	ChefEtat
1	FRA	France	Europe	Europe de l'Ouest	551500.0	59225700	78.8	France	République	Emmanuel Macron

Pour modifier une valeur d'un ou plusieurs attribut(s) pour une ou plusieurs ligne(s) d'une table t selon une condition c, on utilise une requête ayant la forme générale suivante:
UPDATE t SET a1 = e1, a2 = e2, ... WHERE c

Exemple: dans une table Notes:
UPDATE Notes SET Maths=12, Physique=6, WHERE id_eleve= 12;

Faites une petite recherche internet pour mettre à jour les attributs Population et EspeVie de la France.

Remarque: Si la population française a augmenté de 120 000 personnes par exemple, on peut mettre à jour en faisant: SET Population = Population + 120 000