Bases de données SQL

DIU

Sofian MAABOUT

maabout@labri.fr

Au menu

- C’est quoi une Base de Données (BD) ?
- C’est quoi un Système de Gestion de Bases de Données (SGBD) ?
- Le cas particulier des BD’s *relationnelles*
  - Structure
  - Interrogation
    - Algèbre relationnelle (langage théorique)
    - SQL (langage utilisé en pratique)
  - **Aspects “avancés”**
    - **Mise à jour des données**
    - **Définition/création d’une BD**
    - **Contraintes sur les données**
    - **Accès concurrents à une BD**

SQL: insertion

- Ajouter un nouveau livre:

INSERT INTO Livre VALUES(136, « Racines », « Halley », «J'ai Lu»)

On peut aussi

INSERT INTO Livre(Titre, N°Livre, Editeur, NomAuteur) VALUES

(« Racines », 136, « Jai Lu », « Halley »)

SQL: Insertion avec valeur NULL

- Ajouter un nouveau prêt

- Prêt(N°Livre, N°Exemplaire,N°Adhérent,DatePrêt, DateRetour)

INSERT INTO Prêt VALUES (155, 2,445, 15/06/2020)

La date du prêt est enregistrée mais pas la date de retour

<155, 2,445, 15/06/2020, NULL>
Dans ce tuple, la valeur de DateRetour est inconnue.

SQL: Insert avec NULL

- Adhérent(N°Adhérent, Nom, Prénom, Adresse, Mail, Phone)

Ajouter un adhérent qui n'a pas de téléphone

INSERT INTO Adhérent VALUES(33, « Dupond », « Pierre »,
« Bordeaux », DupondAvecD@DupontAvecT.fr)

Ici, la valeur NULL de Phone ne dit pas que le numéro est inconnu, ça dit que pour cet adhérent, l'attribut Phone est inapplicable.

SQL: requête en présence de NULL

- En présence de valeurs NULL dans certains enregistrements, SQL peut évaluer certaines conditions non pas juste à VRAI ou FAUX mais à INCONNU

A	B
10	15
20	
10	17

SELECT count(A)
FROM R
WHERE B>15
Retourne 1 car <10,17> satisfait la condition WHERE

SELECT count(A)
FROM R
WHERE B<=15
Retourne 1 car <10,15> satisfait la condition WHERE

<20, NULL> ne satisfait ni B>15 ni B<=15. Les deux tests sont évalués à Inconnu

SQL: Modification

- L’exemplaire 3 du livre 10 emprunté le 10/05/2020 par l’adhérent 33 vient d’être remis. Il faut donc mettre à jour l’enregistrement correspondant:

```
UPDATE Prêt SET DateRetour=15/06/2020
WHERE N°Exemplaire=3 AND
        N°Livre=10 AND
        N°Adhérent=33 AND
        DatePrêt=10/05/2020
```

SQL: Modification

- L’adhérent 33 qui n’avit pas de téléphone vient d’en avoir un.

UPDATE Adhérent

SET Phone = 06…

WHERE N°Adhérent=33

Noter que dans la condition WHERE, on n’a pas ajouté le nom, prénom … car on estime que N°Adhérent identifie d’une manière unique les adhérents.

SQL: Suppression

- L’adhérent 33 déménage et donc ne veut plus être adhérent. Il faut le supprimer de la base:

```
DELETE FROM Adhérent
WHERE  N°Adhérent = 33
```

SQL: Suppression

- Attention: on ne devrait pas supprimer un adhérent s'il a des prêts en cours (DateRetour est inconnue)

```
DELETE FROM Adhérent
WHERE N°Adhérent = 33 AND
        NOT EXISTS   (SELECT *
                      FROM Prêt
                      WHERE N°Adhérent=33 AND
                       DateRetour IS NULL)
```

NOT EXISTS: teste si un ensemble est vide

IS NULL: teste si un attribut prend la valeur NULL.

On n'écrit jamais Attribut=Null ou Attribut <>NULL.

SQL: Suppression

- Pour s’amuser, supprimons tous les adhérents

DELETE FROM Adhérent

Une suppression sans condition va supprimer le contenu d’une table mais ne va pas supprimer la table.

On pourra donc par la suite faire:

INSERT INTO Adhérent VALUES (…)

SQL: Suppression d'une table

- On veut supprimer la table des adhérents (ce n'est certainement pas utile comme opération)

DROP Table Adhérent

Après l'exécution de cette commande, il n'est plus possible de faire

INSERT INTO Adhérent Values(…)

car la table a disparu.

SQL: Ajout d'une table

- Heureusement, on a un listing papier des adhérents. On va donc pouvoir recréer cette table

```
CREATE TABLE Adhérent (
        N°Adhérent   Integer,
        Nom          Varchar(30),
        Prénom       VarChar(30),
        Adresse      Varchar(100),
        Phone        Char(10),
        Mail         Varchar(100)
)
```

SQL: Identifiant

- Dans la table Adhérent, on veut que deux adhérents ne puissent pas avoir le même numéro: N°Adhérent est un identifiant dans cette table.

```
CREATE TABLE Adhérent (
        N°Adhérent   Integer,
        Nom          Varchar(30),
        Prénom       VarChar(30),
        Adresse      Varchar(100),
        Phone        Char(10),
        Mail         Varchar(100),
        PRIMARY KEY (N°Adhérent)
)
```

Le SGBD va refuser l'insertion d'un nouvel adhérent si son numéro est le même que celui d'un autre adhérent qui est déjà enregistré.

SQL: Identifiant

- Une clé primaire peut être composée de plusieurs attributs

- Ce qui identifie un exemplaire c'est N°Livre et N°Exemplaire

```
CREATE TABLE Exemplaire(
        …
        PRIMARY KEY(N°Livre,N°Exemplaire)
)
```

SQL: Identifiant secondaire

- Dans l'éducation nationale, les employés sont décrits, entre autres, par leur N°SS et leur NUMEN.

```
CREATE  TABLE Enseignant(
        N°SS            Char(15),
        NUMEN           Char(12),
        …
        PRIMARY KEY(NUMEN),
        UNIQUE   (N°SS)
)
```

N°SS est un identifiant secondaire. Le SGBD va s'assurer de son unicité tout comme pour le NUMEN.

Une seule clé primaire par table. On peut avoir plusieurs clés secondaires par table.

SQL: Ajouter/Supprimer une clé a posteriori

- On peut ajouter une clé primaire (ou autres contraintes) après la création d'une table

```
CREATE TABLE Adhérent (
        N°Adhérent   Integer,
        …
        Mail         Varchar(100)
)
ALTER TABLE Adhérent ADD CONSTRAINT C1 PRIMARY KEY(N°Adhérent)
```

C1 est le nom de la contrainte. On peut par la suite la supprimer avec

```
ALTER TABLE Adhérent DROP CONSTRAINT C1
```

SQL: Attribut forcément renseigné

```
CREATE TABLE Livre(
        N°Livre         Integer,
        Titre           Varchar(10) NOT NULL,
        …
)
```

INSERT INTO Livre(N°Livre, NomAuteur,Editeur) VALUES(136, ″A ″, ″B ″)

Sera refusé car le Titre n’est pas renseigné.

SQL: Contrainte de référence ou clé étrangère

- Quand on enregistre un prêt, on veut que le numéro d'adhérent soit celui de quelqu'un qui est déjà enregistré dans la table Adhérent

```
CREATE TABLE Prêt (
        N°Adhérent      Integer,
        …
        DatePrêt        Date DEFAULT SYSDATE(),  //valeur par défaut est la date du jour
        …
        FOREIGN KEY (N°Adhérent) REFERENCES Adhérent(N°Adhérent) )
```

Non seulement on ne pourra pas ajouter un prêt où le numéro d'adhérent n'existe pas mais on ne peut pas non supprimer un adhérent pour lequel on a des prêts enregistrés.

Organisation d'une BD

- On a supposé l'existence d'une BD et on a vu comment l'interroger
- On a supposé l'existence d'une BD et on a vu comment la mettre à jour (Insertion/Suppression/Modification).
- On a vu comment ajouter/supprimer/définir des tables
- On a vu quelques contraintes d'intégrité (retenir les plus importantes: Clé primaire et Clé étrangère)
- Comment décider des tables qui composent la BD et comment décider de la structure (ses attributs) de chaque table?
- Il y a deux grandes lignes/méthodologies qu'on peut adopter
  - Méthodologie conceptuelle en utilisant des techniques telles que Merise et modèle Entité-Association
  - Méthodologie algorithmique basée sur la notion de « dépendance fonctionnelle »

Organisation d'une BD: Problèmes

N°SS	Nom	Ville	Matricule	Modèle	Marque	NbPlaces
1234	Dupont	Bordeaux	234AC218	XT2456	Peugeot	5
2345	Dupond	Pau	234BD517	XT2456	Peugeot	5
1234	Dupont	Bordeaux	456AZ987	YRVS23	Toyota	7
1234	Dupont	Bordeaux	236VC265	AVTY	Nissan	5
2678	Martin	Dax	567AB673	AVTY	Nissan	5

Le nom Dupont et sa ville sont répétés autant de fois qu'il/elle ne possède de voitures

La description du modèle XT24456 est répété autant de fois qu'il n'y a de personnes qui en possèdent un exemplaire

L'information redondante fait gaspiller l'espace de stockage et expose aux anomalies de mise à jour:

Si Dupont part à Bergerac, il faut s'assurer que sa ville soit modifiée partout

Si on veut ajouter une personne qui n'a pas de voiture alors on va forcément utiliser les valeurs NULL

Organisation d'une BD: Décomposition

N°SS	Nom	Ville	Matricule	Modèle	Marque	NbPlaces
1234	Dupont	Bordeaux	234AC218	XT2456	Peugeot	5
2345	Dupond	Pau	234BD517	XT2456	Peugeot	5
1234	Dupont	Bordeaux	456AZ987	YRVS23	Toyota	7
1234	Dupont	Bordeaux	236VC265	AVTY	Nissan	5
2678	Martin	Dax	567AB673	AVTY	Nissan	5

N°SS	Nom	Ville
1234	Dupont	Bordeaux
2345	Dupond	Pau
2678	Martin	Dax

N°SS	Matricule
1234	234AC218
2345	234BD517
1234	456AZ987
1234	236VC265
2678	567AB673

Matricule	Modèle
234AC218	XT2456
234BD517	XT2456
456AZ987	YRVS23
236VC265	AVTY
567AB673	AVTY

Modèle	Marque	NbPlaces
XT2456	Peugeot	5
YRVS23	Toyota	7
AVTY	Nissan	5

A → B A est clé étrangère faisant référence à B

Notion de droits d'accès

- Tous les utilisateurs d'une BD n'ont pas les mêmes droits:
  - Une caissière a le droit d'enregistrer un passage à la caisse
  - Une caissière doit appeler sa chef pour annuler/supprimer une transaction
  - La chef de la caissière ne peut pas savoir le chiffre d'affaires journalier du magasin.
  - Le responsable du magasin ne peut pas savoir le chiffre d'affaire de la chaine en Gironde.
  - …

Notion de droits d'accès

- On distingue 2 grandes familles d'utilisateurs d'une BD
  - Administrateurs d'une BD: tous les droits et très peu nombreux
  - Autres: droits limités/tous sauf les administrateurs
- En pratique, seuls les administrateurs manipulent SQL
- Ce sont les administrateurs qui créent les utilisateurs et leur octroient des droits d'accès
- Les administrateurs créent *des procédures* et autorisent (ou pas) les autres utilisateurs à les exécuter.

Notion de droits

- La bibliothécaire n'est pas censée faire INSERT … quand elle enregistre un nouveau prêt.

```
CREATE Procedure InsertionPret(){
        NA = saisir_num_adhérent();
        NL = saisir_num_livre();
        NE = saisir_num_exemplaire();
        Insert Into Prêt Values(NA, NL, NE, SYSDATE());
}

GRANT EXECUTE ON InsertionPret TO Bibliothécaire;
```

La bibliothécaire en exécutant cette procédure (ex: run InsertionPret()), n'aura qu'à saisir NA, NL et NE pour que l'insertion dans la base soit réalisée. Pas de manipulation de SQL.

Accès concurrents

- Deux personnes qui consultent simultanément le nombre de places disponibles pour un vol.
- Les deux voient qu'il reste deux places (interrogent une BD en réalité)
- Une personne passe commande d'un billet, l'autre de 2 billets.
- Une des deux va se retrouver en surbooking si les deux commandes sont réalisées
- Le SGBD doit gérer ce type de conflits.
- Dans un SGBD, la part du code dévolue à la gestion des accès concurrents représente plus de la moitié

Développement d'applications autour d'une ou utilisant une BD

- La plupart des langages de programmation possèdent des API permettant d'accéder (interroger/modifier/exécuter) à une BD

- Exemple typique: Un site WEB qui permet d'acheter ses billets de cinéma.
  - Le site n'est certainement pas développé en SQL
  - L'achat d'un billet doit être enregistré dans une BD donc forcément utilisation de SQL.

- Une BD est rarement utilisée sans intégration avec une application évoluée (interface graphique/facilité d'utilisation …)

Conclusion

- On a vu essentiellement l'interrogation d'une BD qui existe
- Il est légitime se poser la question: d'où viennent ces données ?
- Il est légitime de se poser la question: pourquoi a-t-on décidé d'organiser la BD de la sorte ?
- On a essayé de donner un aperçu d'autres aspects qui permettent de
  - Situer la place des BD's dans l'informatique de tous les jours
  - L'éventail des fonctionnalités offertes par un SGBD
- Les élèves doivent au moins être conscients que derrière l'application ParcourSup il y a forcément une BD☺
  - Saisir ses vœux c'est insérer un ou des enregistrements
  - Changer d'avis c'est modifier un ou des enregistrements