Cours de bases de données

http ://www.lamsade.dauphine.fr/rigaux/bd

SGBD relationnels
Travaux pratiques

Philippe Rigaux

Table des matières

1 Interrogation avec SQL **2**
1.1 Sélections simples . . . 3
1.2 Jointures . . . 4
1.3 Requêtes imbriquées . . . 4
1.4 Négation . . . 5
1.5 Fonctions de groupe . . . 5

2 Création d'un schéma relationnel **5**
2.1 Création des tables . . . 5
2.2 Insertion de données . . . 6
2.3 Vues . . . 6

3 Programmation **7**
3.1 Procédures stockées . . . 7
3.2 *Triggers* . . . 7
3.3 Applications externes : PHP . . . 7

4 Concurrence d'accès **8**

Tous les travaux pratiques décrits dans ce qui suit peuvent s'effectuer en ligne grâce à l'interface web disponible à l'URL :

`http ://www.lamsade.dauphine.fr/rigaux/bd`

La mise en œuvre est simplissime puisqu'il n'y a rien à installer : vous avez juste besoin d'un ordinateur accédant au Web. Les instructions pour utiliser le système sont disponibles en ligne. Sachez simplement qu'il est possible de pratiquer SQL sans autre formalité. En revanche, pour créer une base et disposer d'un environnement de travail permettant des créations, des modifications et l'écriture d'une application, il faudra fournir quelques informations permettant de créer un compte. Vous accéderez ensuite au système avec un login et un mot de passe.

Il est également possible d'effectuer ces TP dans d'autres conditions : le seul pré-requis est en fait de disposer d'un SGBD relationnel suffisamment avancé, et d'un terminal d'accès permettant d'entrer les commandes en ligne. Voyez dans un tel cas avec votre enseignant quelles sont les opérations d'installation à effectuer.

Les exercices proposés comprennent à peu près toutes les opérations dont la maîtrise est nécessaire à la pratique correcte et efficace un SGBD relationnel, à savoir :

- interrogation avec le langage SQL (section 1, page 1) ;
- insertions et mise à jour diverses et variés ;
- conception et création d'un schéma avec tables, vues, contraintes, index, etc. ;
- mécanisme de concurrence d'accès ;
- accès à la base avec un langage de programmation interne (PL/SQL) ou externe (Java, PHP).

Un projet récapitulant l'ensemble de ces connaissances vous est également proposé.

Il est clair que la présence d'un enseignant pour vous guider est préférable. Si ce n'est pas le cas, vous ne risquez rien à tenter votre chance pour expérimenter et voir si vous vous en sortez ou non. Dans tous les cas, bonne chance !

1 Interrogation avec SQL

Au départ, vous avez accès au schéma et à la base de données « Films », vue et revue en cours, qui est partagée (en lecture) par tout le monde. Voici le script de création de ce schéma (disponible sur le site). Ces commandes doivent maintenant vous être familières (sinon relisez les chapitres correspondant).

Exemple 1 *SchemaFilms.sql : Le script de création du schéma*

```
/*
   Commandes de création de la base Films,  testé avec MySQL et PostgreSQL.
   Pour Oracle, il suffit de remplacer le type TEXT par le type LONG dans
   la table Film.
   Philippe Rigaux, 2004
*/

/* Destruction eventuelle des tables existantes */

DROP TABLE  Notation;
DROP TABLE  Role;
DROP TABLE  Film;
DROP TABLE Artiste;
DROP TABLE  Internaute;
DROP TABLE  Pays;
DROP TABLE  Genre;

/* Creation des tables                          */

CREATE TABLE Internaute (email VARCHAR (40) NOT NULL,
                         nom VARCHAR (30) NOT NULL ,
```

```
                        prenom VARCHAR (30) NOT NULL,
                        region VARCHAR (30),
                        CONSTRAINT PKInternaute PRIMARY KEY (email));

CREATE TABLE Pays (code    VARCHAR(4) NOT NULL,
                   nom  VARCHAR (30) DEFAULT 'Inconnu' NOT NULL,
                   langue VARCHAR (30) NOT NULL,
                   CONSTRAINT PKPays PRIMARY KEY (code));

CREATE TABLE Artiste  (idArtiste INTEGER NOT NULL,
                       nom VARCHAR (30) NOT NULL,
                       prenom VARCHAR (30) NOT NULL,
                       anneeNaiss INTEGER,
                       CONSTRAINT PKArtiste PRIMARY KEY (idArtiste),
                       CONSTRAINT UniqueNomArtiste UNIQUE (nom, prenom));

CREATE TABLE Film  (idFilm INTEGER NOT NULL,
                    titre    VARCHAR (50) NOT NULL,
                    annee    INTEGER NOT NULL,
                    idMES    INTEGER,
                    genre VARCHAR (20) NOT NULL,
                    /* Remplacer TEXT par LONG pour ORACLE */
                    resume      TEXT,
                    codePays    VARCHAR (4),
                    CONSTRAINT PKFilm PRIMARY KEY (idFilm),
                    FOREIGN KEY (idMES) REFERENCES Artiste,
                    FOREIGN KEY (codePays) REFERENCES Pays);

CREATE TABLE Notation (idFilm INTEGER NOT NULL,
                       email  VARCHAR (40) NOT NULL,
                       note  INTEGER NOT NULL,
                       CONSTRAINT PKNotation PRIMARY KEY (idFilm, email));

CREATE TABLE Role (idFilm  INTEGER NOT NULL,
                   idActeur INTEGER NOT NULL,
                   nomRole  VARCHAR(30),
                   CONSTRAINT PKRole PRIMARY KEY (idActeur,idFilm),
                   FOREIGN KEY (idFilm) REFERENCES Film,
                   FOREIGN KEY (idActeur) REFERENCES Artiste);

CREATE TABLE Genre (code    VARCHAR (20) NOT NULL,
                    CONSTRAINT PKGenre PRIMARY KEY (code));
```

Vous pouvez remarquer que l'ordre de création des tables respecte le référencement entre `PRIMARY KEY` et `FOREIGN KEY`. Les tables qui sont référencées par cette dernière clause doivent être créées *avant* celles qui les référencent. Par exemple la table *Artiste* est créée avant la table *Film* à cause de la clé étrangère `idMES`. C'est en revanche l'ordre inverse qui est suivi pour les commandes `DROP` : on ne peut pas détruire une table qui est référencée par une commande `FOREIGN KEY`. Notez qu'en principe on ne place pas les commandes `DROP` dans un script de création puisqu'on ne souhaite pas prendre le risque de détruire des données existantes. Comme il s'agit ici d'une base de test, la situation est différente.

La base est disponible sur le site et contient un échantillon de films avec leur metteur en scène, leurs acteurs et les notations de quelques internautes. À vous de jouer : il faut concevoir, saisir et exécuter les ordres SQL correspondant aux requêtes qui suivent.

1.1 Sélections simples

1. Tous les titres de films.

2. Nom et prénom des internautes auvergnats.
3. Titre et année de tous les drames, triés par année ascendante. Donnez ensuite le tri par année descendante.
4. Nom et année de naissance des artistes nés avant 1950.
5. Titre et année de tous les films parus entre 1960 et 1980
6. Tous les genres de films (éliminez les doublons).
7. Titre, genre et résumé des films qui sont soit des drames, soit des westerns (utilisez la construction `IN`), et dont le résumé contient la chaîne de caractères « vie ».
8. Les artistes dont le nom commence par 'H' (commande `LIKE`).
9. Quels sont les acteurs dont on ignore l'année de naissance ? (Attention : cela signifie que la valeur est absente).
10. Prénom, nom et âge de chaque artiste (NB : l'âge est la différence entre l'année courante et l'année de naissance). Nommez `âge` la colonne obtenue (commande `AS`).

1.2 Jointures

1. Qui joué le rôle de *Morpheus* (nom et prénom) ?
2. Qui est le réalisateur de *Alien* ?
3. Prénom et nom des internautes qui ont donné une note de 4 à un film (donner aussi le titre du film).
4. Quels acteurs ont joué quel rôle dans le film *Vertigo* ?
5. Films dont le réalisateur est Tim Burton, et un des acteurs est Johnny Depp.
6. Titre des films dans lesquels a joué Bruce Willis. Donner aussi le nom du rôle.
7. Quel metteur en scène a tourné dans ses propres films ? Donner le nom, le rôle et le titre des films.
8. Quel metteur en scène a tourné en tant qu'acteur (mais pas dans son propre film) ? Donner le nom, le rôle et le titre des films où le metteur en scène a joué.
9. Dans quels films le metteur en scène a-t-il le même prénom que l'un des interprètes ? (titre, nom du metteur en scène, nom de l'interprète). Le metteur en scène et l'interprète ne doivent pas être la même personne.

1.3 Requêtes imbriquées

Les requêtes suivantes peuvent s'exprimer avec une imbrication des clauses `SELECT`, mais on peut également utiliser des jointures « à plat ». Si le cœur vous en dit, essayez les deux versions.

1. Donnez les nom et prénom des artistes qui on mis en scène un film.
2. Donnez le titre et année des films qui ont le même genre que *Matrix*.
3. Donnez le nom des internautes qui ont noté le film *Alien*. Donnez également la note.

1.4 Négation

Là encore, il existe souvent plusieurs manières d'exprimer la même requête.

1. Les films sans rôle.
2. Nom et prénom des acteurs qui n'ont jamais mis en scène de film.
3. Les internautes qui n'ont pas noté de film paru en 1999.

1.5 Fonctions de groupe

1. Quelle est le nombre de films notés par l'internaute *rigaux@cnam.fr*, quelle est la moyenne des notes données, la note minimale et la note maximale ?
2. Combien de fois Bruce Willis a-t-il joué le role de McClane ?
3. Année du film le plus ancien et du film le plus récent.
4. id, Nom et prénom des réalisateurs, et nombre de films qu'ils ont tournés.

2 Création d'un schéma relationnel

Il s'agit de définir un schéma de base de données, d'y intégrer des contraintes, des vues et d'y insérer quelques informations. Vérifiez le comportement des contraintes et des vues en essayant de les mettre en défaut.

2.1 Création des tables

Créez les tables du schéma 'Agence de voyages', vues en cours, et rappelées ci-dessous.
- Station (**nomStation**, capacité, lieu, région, tarif)
- Activite (***nomStation***, **libellé**, prix)
- Client (**id**, nom, prénom, ville, région, solde)
- Sejour (***id***, ***station***, **début**[1], nbPlaces)

Attention à bien définir les clés primaires et étrangères. Voici les autres contraintes portant sur ces tables.

1. Les données `capacité, lieu, nom, ville, solde` et `nbPlaces` doivent toujours être connues.
2. Les montants (prix, tarif et solde) ont une valeur par défaut à 0.
3. Il ne peut pas y avoir deux stations dans le même lieu et la même région.
4. Les régions autorisées sont : 'Ocean Indien', 'Antilles', 'Europe', 'Ameriques' et 'Extreme Orient'.
5. La destruction d'une station doit entraîner la destruction de ses activités et de ses séjours.
6. Le prix d'une activité doit être inférieur au tarif de la station et supérieur à 0.
7. (♣)Pour une date de début donnée, le nombre total de places réservées dans une station doit être inférieur à la capacité de la station.

Conseil : donnez des noms à vos contraintes `PRIMARY KEY`, `FOREIGN KEY` et `CHECK` avec la clause `CONSTRAINT`.

[1] Utiliser le type `DATE`

Station

NomStation	Capacité	Lieu	Région	Tarif
Venusa	350	Guadeloupe	Antilles	1200

Activite

NomStation	Libellé	Prix
Venusa	Voile	150
Venusa	Plongée	120

Client

id	nom	prénom	ville	région	solde
10	Fogg	Phileas	Londres	Europe	12465
20	Pascal	Blaise	Paris	Europe	6763
30	Kerouac	Jack	New York	Amérique	9812

Sejour

idClient	station	début	nbPlaces
20	Venusa	03-08-2003	4

FIG. 1 – La base 'Agence'

2.2 Insertion de données

Insérez dans la base les données de la figure 1 avec des ordres `INSERT`. Attention, l'ordre des `INSERT` est important (pourquoi ?).

Vous pouvez ensuite tester les contraintes avec quelques ordres SQL. Par exemple : détruisez la station et vérifiez que les activités ont disparu ; insérez une autre station en (Guadeloupe, Antilles) ; insérez une station dans une région 'Nullepart', etc.

2.3 Vues

Maintenant il faut créer des vues et tester l'interrogation et la mise à jour à travers ces vues.

1. Créez les vues suivantes sur le schéma précédent.
   (a) Une vue *ActivitesModiques (Station, Activite)* donnant le nom des stations et des activités dont le prix est inférieur à 140 FF. Toute ligne insérée dans cette vue doit apparaître dans la vue ensuite.
   (b) Une vue *ActivitesCheres*, de même schéma, avec prix supérieur à 140 FF, et la même contrainte d'insertion.
   (c) Une vue *StationDollars (Nom, Capacite, Lieu, TarifDollar)* donnant le nom d'une station, sa capacité, le lieu et le tarif en dollars (metez le taux de conversion « en dur », ou bien – mieux – créez une table stockant le taux de conversion).
   (d) Une vue *Tarifs (Station, Tarif, OptionMin, OptionMax)* donnant, pour chaque station, le tarif et les prix min et max des activités.
   (e) Une vue *Reservation (nomStation, PlacesReservees)* donnant, par station et date de début de séjour, le nombre de places réservées.
2. Consultez ensuite le contenu de ces vues. Vous pouvez insérez quelques lignes supplémentaires dans les tables et constater qu'elles sont prises en compte dans les vues.
3. Dans quelles vues peut-on insérer, détruire et mettre à jour ? Essayez les opérations suivantes :
   (a) Insérez une activité 'Kayac' pour la station 'Venusa' dans *ActivitesCheres* et *ActivitesModiques*. Le contrôle sur l'insertion est-il utile dans ce cas ?

(b) Peut-on insérer dans *StationEuro* ? Sous quelle condition ? Faites l'essai.

(c) Détruisez une ligne de *StationEuro*.

3 Programmation

Les exercices qui suivent permettent de se familiariser avec les langages de programmation interne (PL/SQL) et externe (Java/JDBC, PHP) permettant de manipuler une base de données de manière procédurale. Des exemples sont fournis sur le site, qui doivent vous permettre de créer vos propres procédures, fonctions et programmes.

3.1 Procédures stockées

Le langage utilisé est le PL/SQL version PostgreSQL. Quelques modifications mineures suffisent à transcrire pour le PL/SQL d'Oracle.

1. Créer une fonction *NomClient* qui prend en entrée l'id d'un client et qui renvoie une chaîne contenant le prénom et le nom du client (voir l'exemple *réalisateur.plsql*).
2. Créer une fonction *Activités* qui prend en entrée le nom du station et produit une chaîne de caractères contenant l'énumération des activités de la station (par exemple, "`Ski`, `Yoga`, `Massage`").
3. Créer ensuite une vue qui affiche les stations, avec un attribut supplémentaire donnant la liste des activités (par appel à la fonction bien sûr).
4. Créer une fonction *Actualiser* qui prend en entrée un pourcentage et le nom d'une station, et augmente le tarif de la station et le prix de chacune de ses activités du pourcentage indiqué.

3.2 *Triggers*

1. Implantez par un *trigger* la règle suivante : si le prix d'une activité baisse, alors le tarif de la station doit augmenter de la différence.

   Indication : le *trigger* doit se déclencher sur une modification, et tester pour chaque ligne que la nouvelle valeur est plus grande que l'ancienne. Si ce n'est pas le cas, faire un `UPDATE` de la station pour ajouter la différence entre l'ancienne et la nouvelle valeur.
2. Faites l'expérience : diminuez le prix d'une activité, et regardez ce qui se passe pour la station.
3. On veut disposer de l'information `nbActivites` dans la table *Station*. Pour cela :
   (a) Ajoutez la colonne *nbActivites* avec pour valeur par défaut 0.
   (b) Créez un *trigger* qui maintient cette information.
4. Interdisez par un *trigger* l'insertion d'une ligne dans la table *Séjour* si le solde du client est inférieur au nombre de places multiplié par le tarif de la station.

3.3 Applications externes : PHP

Il n'y a aucune limite (enfin, presque), à ce que l'on peut faire avec une programmation HTML/PHP/PostgreSQL (ou n'importe quelle autre SGBD). Pour commencer vous devez installer les trois scripts vus en cours qui montrent les principales techniques (connexion, exécution d'une requête, interrogation par formulaire). Récupérez les fichiers sur le site (ils sont dans une archive *pgphp.tar* disponible dans la page des exemples). Vous devez ensuite effectuer les opérations suivantes :

1. Créer les tables *FilmComplet* et *FilmSimple* avec les deux scripts *FilmSimple.sql* et *FilmCOmplet.sql*
2. Insérer quelques lignes dans la table *FilmSimple* avec le script *InsFilmSimple.sql*
3. Modifier les paramètres de connexion dans *Connect.php*

4. *ExPGSQL1.php* interroge et affiche le contenu de *FilmSimple*.
5. *ExForm3.html* affiche un formulaire pour rechercher des données en lançant le script *ExPGSQL2.php*
6. *ExForm4.html* affiche un formulaire pour insérer, modifier oui détruire des données en lançant le script *ExPGSQL3.php*

Vous trouverez beaucoup plus de scripts (pour MySQL, à adapter à PostgreSQL) sur le site du livre *Pratique de MySQL et PHP*, à l'URL *http ://www.lamsade.dauphine.fr/rigaux/mysqlphp*.

Voici quelques suggestions pour des scripts PHP sur la base *Station*.

1. Récupérez les scripts PHP donnés en exemple, et travaillant sur la table *FilmSimple*, et adaptez-les pour saisir et modifier les informations de la table *Client*.
2. Faites un formulaire de saisie d'un séjour, en proposant la liste des stations et la liste des clients dans un menu déroulant.

Ces exercices sont réalisables sur le site de TP en ligne.

4 Concurrence d'accès

On va maintenant étudier le comportement concret d'un SGBD en cas d'accès concurrents à la même ressource. Pour celà on va simuler l'exécution concurrente de programmes à l'aide du petit ensemble de lectures/écritures sur la base "Agence de voyage" créé dans le premier exercice : modification du solde de certains clients et sélection des clients. Créez 4 fichiers, nommés `SEL.sql, MAJpas.sql, MAJfog.sql et MAJker.sql` et entrez-y les commandes suivantes :

1. `SEL.sql`

```
PROMPT 'Affichage des clients =>';
SELECT * FROM client;
```

2. `MAJpas.sql`

```
PROMPT 'Augmentation du client Pascal =>';
UPDATE client  SET solde = solde + 1000
WHERE  client = 'Pascal';
```

3. `MAJfog.sql`

```
PROMPT 'Augmentation du client Fogg =>';
UPDATE client SET solde = solde + 1000
WHERE  client = 'Fogg';
```

4. `MAJker.sql`

```
PROMPT 'Augmentation du client Kerouac =>';
UPDATE client SET solde = solde + 1000
WHERE  client = 'Kerouac';
```

Ensuite, ouvrez deux fenêtres et lancez SQLPLUS dans chacune : chaque session est considérée par ORACLE comme un utilisateur, et on a donc 2 utilisateurs, nommés 1 et 2, en situation de concurrence. Dans tout ce qui suit, on note $INSTR_i$ l'exécution de l'instruction $INSTR$ par l'utilisateur i. Par exemple $MAJker_1$ corespond à l'exécution du fichier `MAJker` dans la première fenêtre par la commande `@MAJker`. On note de même ROL_i et COM_i l'exécution des commandes `rollback ;` et `commit ;` dans la fenêtre i.

Questions Executez les séquences d'instruction décrites ci-dessous. Dans chacun des cas, expliquez ce qui se passe.

1. Première expérience : l'utilisateur 1 effectue des mises-à-jour, tandis que l'utilisateur 2 ne fait que des sélections. Que constate-t-on ?

   $$SEL_1, SEL_2, MAJker_1, SEL_2, MAJpas, SEL_2, ROL_1, SEL_2.$$

2. idem, mais avec des $commit$.

   $$SEL_1, SEL_2, MAJker_1, SEL_2, COM_1, SEL_2, MAJker_1, SEl_2, COM_1, SEL_2.$$

3. Maintenant les deux utilisateurs effectuent des MAJ simultanées.

   $$SEL_1, SEL_2, MAJker_1, MAJpas_2, SEl_1, SEL_2, MAJfog_1, MAJfog_2, SEL_1, COM_1, COM_2.$$

   Un bloquage est apparu. Pourquoi ?

4. Idem, avec un ordre des opérations qui diffère d'un utilisateur à l'autre.

   $$SEL_1, SEL_2, MAJker_1, MAJpas_2, SEl_1, SEL_2, MAJpas_1, MAJker_2 ROL_1 ROL_2.$$

   Que constate-t-on ?

5. (♣)En fait, ORACLE pratique une verrouilage à deux phases assez libéral, qui ne garantit pas la sérialisabilité : aucun verrrou n'est placé sur une ligne lors d'une lecture. L'utilisateur peut verrouiller explicitement en ajoutant la clause `FOR UPDATE`. Exemple :

   ```
   SELECT * FROM client FOR UPDATE;
   ```

   Chaque ligne sélectionnée est alors verrouillée. Refaites les expériences précédentes avec un verrouillage explicite des lignes que vous voulez modifier.

6. Expérimentez les exécutions précédentes en spécifiant le mode suivant :

   ```
   SET TRANSACTION ISOLATION LEVEL SERIALIZABLE
   ```