Base De Données

Version Enseignant

Eric BERTHOMIER
eric.berthomier@free.fr

24 février 2024

Table des matières

1 Introduction **5**
- 1.1 Historique . . . 5
- 1.2 La pédagogie . . . 5
- 1.3 Les contraintes . . . 5
- 1.4 Les programmes . . . 5
- 1.5 Fichiers d'accompagnement . . . 5
- 1.6 Rappel de convention . . . 6
- 1.7 Environnement Linux . . . 6
- 1.8 Environnement utilisateur Windows . . . 7
  - 1.8.1 Putty . . . 7
  - 1.8.2 WinScp . . . 8

I Étude de cas - Une Pizzéria **9**

2 Mise en appétit... **11**

3 Premières Pizzas **13**
- 3.1 Première étape : la ou les pizzas . . . 13
  - 3.1.1 Ma future carte 13
  - 3.1.2 Classement . . . 13
  - 3.1.3 Composition . . . 13
- 3.2 Nos premières tables de données . . . 13
  - 3.2.1 Création de la base et d'un compte associé . . . 13
  - 3.2.2 Utilisation d'un fichier sql . . . 14
  - 3.2.3 Connexion à une base de données . . . 14
  - 3.2.4 Création des 2 tables de données . . . 15
  - 3.2.5 Garniture de la table . . . 15
- 3.3 On met le couvert . . . 18
  - 3.3.1 Affichage des données en Python . . . 18
- 3.4 Hep garçon, s'il vous plait, auriez-vous ... ? . . . 20
- 3.5 Garniture + Pâte = Pizza . . . 22
  - 3.5.1 Une première Pizza un peu lourde . . . 22
  - 3.5.2 Une Pizza allégée . . . 24
  - 3.5.3 Une Pizza poids forme . . . 27
- 3.6 Bilan . . . 31
- 3.7 Sauvegarde . . . 32

4 Pizzaïolo **33**
- 4.1 Vite . . . 33
- 4.2 Un collègue bien impatient 33
  - 4.2.1 Dictionnaire de données . . . 34
- 4.3 Sauvegarde . . . 38

5 Devoir à la maison **39**
5.1 Intitulé du devoir . . . 39
5.1.1 Livrable attendu . . . 39
5.2 Consignes . . . 39
5.3 Précisions . . . 39

6 Pizza Requêtes **41**
6.1 Préambule . . . 41
6.2 Un plat toujours chaud . . . 41
6.2.1 Exemple . . . 41
6.3 Warnings . . . 41
6.4 Ajout de données . . . 41
6.4.1 Modification de la table Garnitures . . . 42
6.4.2 Marche à suivre . . . 42
6.4.3 Prix des pâtes . . . 43
6.5 Requêtes simples . . . 44
6.5.1 Interrogation . . . 44
6.5.2 Calcul . . . 45
6.5.3 Mise à jour . . . 45
6.6 Requêtes temporelles . . . 46
6.7 Jointures . . . 46
6.7.1 Exemple commenté . . . 46
6.8 Exercices sur les Jointures . . . 48

7 Pizzas améliorées **49**
7.1 Clé primaire . . . 49
7.2 Gestion des contraintes d'unicité . . . 49
7.2.1 À l'aide du mot clé unique . . . 49
7.2.2 À l'aide d'une clé primaire . . . 50
7.3 Clé étrangère . . . 50
7.4 Miam . . . 51

II Théorie des Bases de données **53**

8 Cours **55**
8.1 Introduction . . . 55
8.2 SQL & MySQL . . . 55
8.3 Architecture . . . 55
8.4 Objets . . . 55
8.5 Bases de Données . . . 56
8.6 Notre Base de Données . . . 56
8.7 Clés primaires et étrangères . . . 56
8.8 LMD - Verbes . . . 57
8.8.1 LMD - SELECT . . . 57
8.8.2 LMD - Jointure . . . 59
8.8.3 LMD - Insert . . . 60
8.8.4 LMD - Update . . . 60
8.8.5 LMD - Delete . . . 60
8.9 LDD, verbes . . . 61
8.9.1 LDD, Create Table . . . 61
8.9.2 LDD, Alter Table . . . 62
8.9.3 LDD, Drop Table . . . 62
8.10 Quelques fonctions usuelles . . . 62
8.11 Index . . . 63
8.12 Colonnes auto_increment . . . 65
8.12.1 Le dernier élément . . . 65
8.13 Index Fulltext . . . 65

III Étude de cas - Un Zoo 67

9 Zoo : Présentation du problème 69

10 Zoo : Schéma relationnel 71
10.1 Introduction . . . 71
10.2 Tables . . . 71
10.3 Suppression des contraintes de clés étrangères . . . 76
10.3.1 Identification des contraintes . . . 76
10.3.2 Suppression d'une contrainte . . . 76
10.4 Compréhension des relations . . . 77
10.5 Requêtes . . . 78

11 Zoo : Évolution des relations 81
11.1 Maladies . . . 81
11.2 Horodatage . . . 82

IV Sécurité 83

12 Les droits des utilisateurs 85
12.1 Préambule . . . 85
12.2 Droits d'accès . . . 85
12.2.1 Exemple . . . 87
12.3 grant . . . 87
12.3.1 Exemple . . . 88
12.4 revoke . . . 88
12.5 set password . . . 88
12.5.1 Exemple . . . 88
12.6 show grants . . . 88
12.6.1 Exemple . . . 88

13 Bases de Données et Sécurité 89
13.1 Firewall et Serveurs . . . 89

V Pour aller plus loin 91

14 Triggers - Déclencheurs 93
14.1 Source . . . 93
14.2 Trigger / Déclencheur . . . 93
14.3 Utilisation simple . . . 93
14.3.1 Exemple utilisé pour l'illustration . . . 93
14.3.2 Abandon de panier . . . 94
14.3.3 Traçabilité . . . 94
14.4 Utilisation évoluée . . . 95
14.4.1 Exercices . . . 96
14.5 Triggers en Cascade . . . 96
14.6 Pizza . . . 98
14.6.1 Suppression d'un type de Pâte . . . 98
14.7 Retour au Zoo . . . 98

15 Procédures Stockées 99
15.1 Source . . . 99
15.2 Définition d'une procédure stockée . . . 99
15.3 Inconvénient d'une procédure stockée . . . 99
15.4 Hello World ! . . . 99
15.5 Mise en application - Base de données Pizzas . . . 99
15.6 Exemple d'usage des paramétres dans une procèdure stockèe . . . 100
15.7 Usage de variables locales . . . 100

15.7.1 Étendue de visibilitè d'une variable . . . 101
15.7.2 Prix total . . . 102

16 PHP / MySQL **103**
16.1 Au commencement : le HTML . . . 103
16.2 CGI . . . 105
16.2.1 Exemples . . . 105
16.3 PHP . . . 106
16.4 Les formulaires . . . 106
16.4.1 Formulaire HTML . . . 106
16.4.2 Formulaire PHP . . . 107
16.5 Connexion à une base de données en PHP . . . 108
16.6 Exercice . . . 108
16.6.1 Création de la base et de l'utilisateur associé . . . 109
16.6.2 Création de la table `users` . . . 109
16.6.3 Remplissage de la table `users` . . . 109
16.6.4 PHP - Lecture de la base . . . 109
16.6.5 HTML - Formulaire . . . 110
16.6.6 PHP - Validation Login / Mdp . . . 110
16.7 Sécurité : SQLInjection . . . 111
16.7.1 SQLInjection : À vous de jouer 111
16.8 Retour sur nos pizzas . . . 112
16.8.1 HTML . . . 112
16.8.2 Exercice . . . 112
16.9 Formulaires rapides . . . 114

VI Annexes **115**

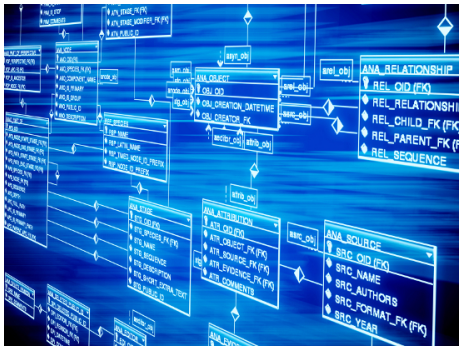
17 Les Accents **117**
17.1 La table . . . 117
17.2 Le fichier d'entrée . . . 117
17.3 Le programme . . . 117
17.4 Le final . . . 118

18 phpMyAdmin **119**
18.1 Introduction . . . 119
18.2 Menu de gauche . . . 119
18.3 Menu principal . . . 120
18.4 Astuces . . . 120
18.4.1 Utilisateurs . . . 120
18.4.2 Renommage . . . 121
18.4.3 Export . . . 121
18.4.4 Sauvegarde . . . 122
18.4.5 Concepteur . . . 122

19 MySQL Workbench **125**
19.1 Description - Extrait du site . . . 125
19.2 Installation . . . 125
19.3 Mode opératoire standard . . . 125
19.4 Ingénierie inverse . . . 126

Chapitre 1

Introduction



1.1 Historique

Après avoir dormi pendant plusieurs années, ce cours revit pour former apprentis et stagiaires, Python 2 n'est plus très utilisé, apparaît donc Python 3 et avec ceci une relecture et une correction de trop nombreuses coquilles de français.

Cette version a été corrigée pour la partie "Pizza"

1.2 La pédagogie

Le cheminement pédagogique utilisé est celui de l'erreur vers les règles de l'art.
Le premier projet partira donc d'une conception de base de données non réfléchie voir même erronée. L'usage de cette dernière démontrera la problématique engendrée par ce schema et ainsi nous conduira vers la raison d'être de tous les éléments qui font la force d'une base de données.

1.3 Les contraintes

Vous remarquerez que les données utilisées sont notées sans accents.
Nous avons voulu en effet simplifier la tâche d'apprentissage sans utiliser les accents qui mettent en œuvre de nombreux éléments :

- la console
- l'encodage du navigateur
- l'encodage des tables
- l'encodage des requêtes dans le cas de programmation externe

Cependant, des informations relatives à cette problématique vous sont données dans la partie annexe (chapitre 17 page 117) de ce support.

1.4 Les programmes

Nous sommes dans un cours de Bases de Données, les programmes Python / PHP ou autres sont volontairement non optimisés afin d'en permettre une lecture simplifiée.

1.5 Fichiers d'accompagnement

Dans le répertoire `Fichiers_TP`, vous devez trouver les éléments suivants :

Fichier	Contenu
`garniture.txt`	Liste des garnitures de pizza
`prix_garniture.csv`	Liste des prix des garnitures de pizza
`demdata.csv`	Fichier de démonstration
`zoo.sql`	Base de données du client pour le zoo
`panier.sql`	Base de données pour les web marchand

Les logins et mot de passe de la machine virtuelle Lubuntu fournie sont `tux` / `tux`. Les logins et mot de passe de la Mysql sont `root` / `droopy`.

1.6 Rappel de convention

— `eric@Tuxie: $` indique un prompt Linux
— `mysql>` indique un prompt SQL

1.7 Environnement Linux

L'environnement est constitué de :
— Langage de programmation
  — Python 2 ou 3
— Interfaces de développement
  — phpmyadmin
  — mysqlworkbench
— Interfaces de saisie (au choix)
  — scite (équivalent de NotePad++)
  — ERIC (of course)
  — Editra
— Consoles
  — Terminator
  — Tilda

Pour permettre l'exécution des programmes python, il est nécessaire d'installer la librairie MySQL associée à Python :
— Python2 : `sudo apt install python-mysqldb`
— Python3 : `sudo apt install python3-mysql.connector`

Pour éviter de devoir manipuler des mots de passe de qualité :

```
mysql> UNINSTALL COMPONENT 'file://component_validate_password';
```

Il est possible de valider le fonctionnement de l'interface avec ce petit programme :

Listing 1.1 – `affichage_user.py`

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
import MySQLdb as mdb
import sys

conn = mdb.connect( host="localhost", # votre hôte, normalement le serveur local
                user="root", # utilisateur
                passwd="droopy", # mot de passe
                db="mysql") # nom de la base de données

# Il est nécessaire de créer un objet de type Curseur pour réaliser des requêtes
curs = conn.cursor()

curs.execute("SELECT * FROM user")

rows = curs.fetchall()

for row in rows:
   print row[1]
```

```
curs.close()
conn.close()
```

Listing 1.2 – `affichage_user3.py`

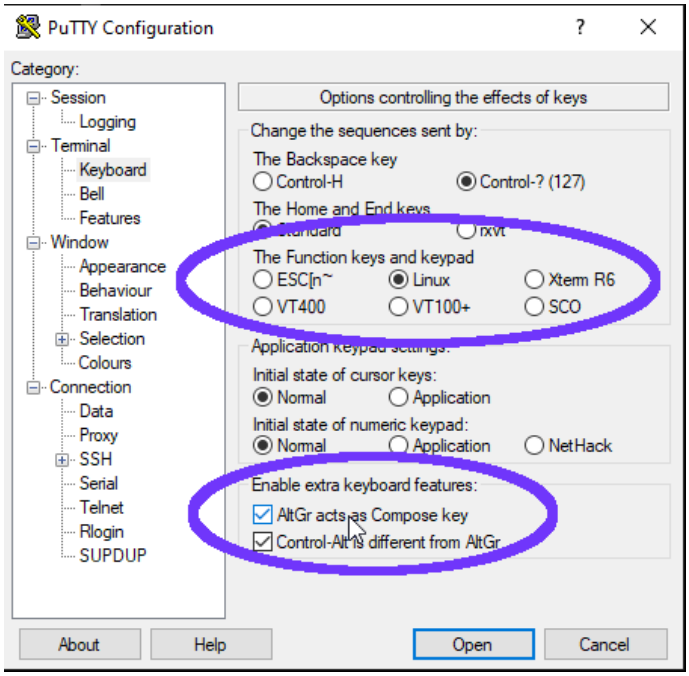
```
#!/usr/bin/python3
# -*- coding: utf-8 -*-

import sys
import mysql.connector
conn = mysql.connector.connect(
user='root',
password='droopy',
host='localhost',
database='mysql'
)

# Traitement des requêtes aussitôt
conn.autocommit = True

# Il est nécessaire de créer un objet de type Curseur pour réaliser des requêtes
curs = conn.cursor()

curs.execute("SELECT * FROM user")

rows = curs.fetchall()

for row in rows:
   print (row[1])

curs.close()
conn.close()
```

1.8 Environnement utilisateur Windows

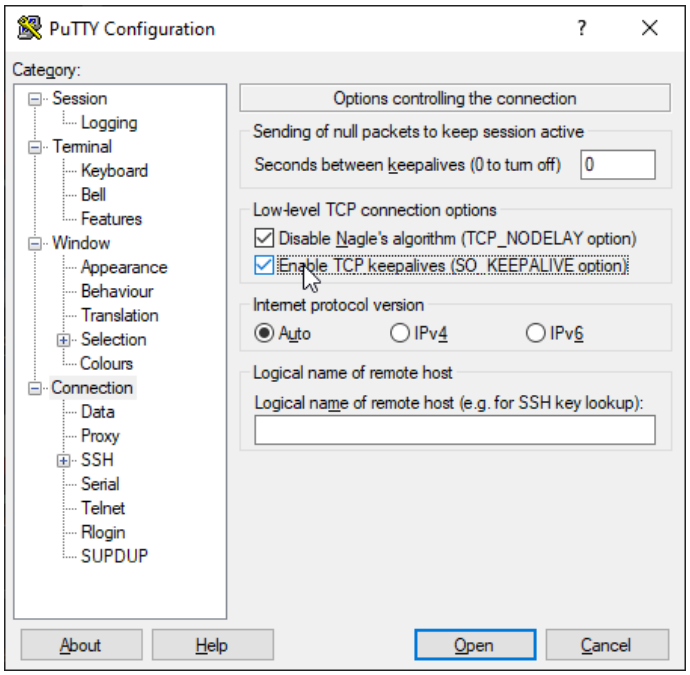
Il est possible de travailler au travers d'un poste de travail Windows en se connectant sur un serveur Linux. Voici cependant quelques astuces pour le bon paramétrage des 2 applications qui seront nécessaires.

1.8.1 Putty

Prise en compte du clavier

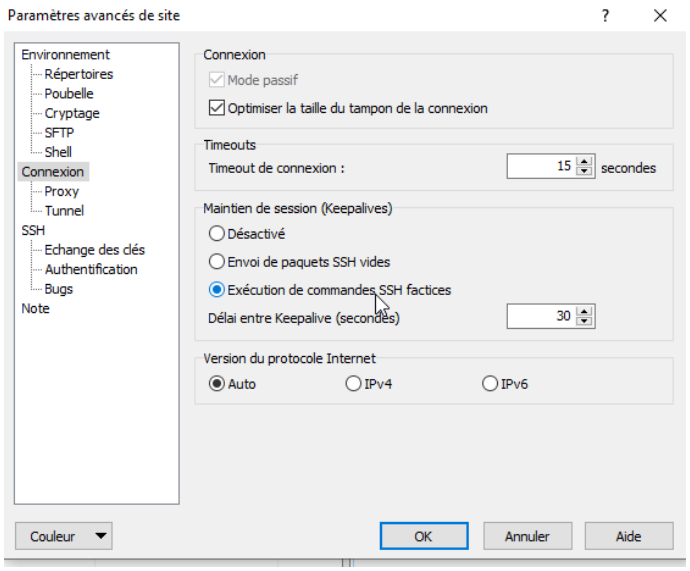


Maintenir la connexion réseau



1.8.2 WinScp

De la même façon que Putty, il est nécessaire d'empêcher la déconnexion réseau pour inactivité, ceci se fait de la façon suivante :



Première partie

Étude de cas - Une Pizzéria

Chapitre 2

Mise en appétit...

Une chaîne de Pizzéria souhaite mettre en place un système permettant à un client de composer sa pizza via Internet mais aussi de permettre au pizzaïolo de disposer des ingrédients sur son plan de travail dès la commande validée.
Il vous adresse donc cette demande qui met en œuvre deux grands éléments :

- une interface de composition des pizzas
- une gestion d'automates pour l'approvisionnement des ingrédients

Chapitre 3

Premières Pizzas

3.1 Première étape : la ou les pizzas

3.1.1 Ma future carte ...

Rechercher ou créer une dizaine de recettes de pizzas sachant que la chaîne propose des pizzas salées mais aussi sucrées avec différentes pâtes. On ne prendra pas en compte la réalisation de la pizza qui relève du métier du pizzaïolo mais uniquement des éléments dont il aura besoin pour la composer.

3.1.2 Classement

Classer les différents éléments trouvés sous forme de groupe.

Exemple

✓ **Type de pâtes :** pâte à pain, pâte complète, pâte à l'eau

Les pâtes

- Classique
- Aux Herbes
- Complète
- Pomme de Terre
- Au Lait
- Farine de Maïs

Les garnitures

- Tomates
- Champignons
- Gruyère
- Merguez
- Crème Fraiche
- Brocolis
- Oignons
- Fromage de chèvre frais
- Courgettes
- Mozzarella
- Olives
- Fontina (fromage italien)
- Parmesan
- Aubergines
- Poivrons rouges
- Poivrons jaunes
- Poivrons verts
- Câpres
- Filet de truite
- Pomme de terre
- Chicorées rouges
- Bacon
- Ananas
- Gorgonzola
- Cerneaux de noix
- Jambon
- Asperges
- Epinards
- Oeuf
- Anchois
- Pomme
- Poire
- Chocolat
- Crevettes
- Raisins
- Banane
- Pruneaux
- Abricots Secs
- Fruits Rouges
- Noix de coco
- Ricotta
- Orange
- Saint Jacques
- Porc Braisé
- Blanc de Poulet
- Ail
- Thon
- Moules
- Palourdes
- Roquette
- Piments
- Miel

3.1.3 Composition

Composer trois nouvelles pizzas en utilisant les différents éléments des pizzas précédemment créés.

3.2 Nos premières tables de données

3.2.1 Création de la base et d'un compte associé

Nous allons insérer tous ces éléments dans les tables d'une base de données. Le premier type de données est la "Pâte" et la seconde la "Garniture".

Nous allons donc créer une base de données "FirstPizza".

Listing 3.1 – `affichage_user3.py`

```
mysql -u root -p
mysql> create database FirstPizza;
```

Rester en tant que superutilisateur de la base de données est source d'erreurs et de grosses bêtises.
Nous allons donc créer un utilisateur puis lui affecter des droits uniquement sur la base de données que nous venons de créer.
Créer l'utilisateur :

Listing 3.2 – `affichage_user3.py`

```
mysql> CREATE USER pizza IDENTIFIED BY 'pizzamdp';
```

Puis lui donner les droits d'Administrateur sur cette base.

Listing 3.3 – `affichage_user3.py`

```
mysql> USE FirstPizza;
mysql> GRANT ALL PRIVILEGES ON FirstPizza.* TO 'pizza'@'%' WITH GRANT OPTION;
mysql> FLUSH PRIVILEGES;
```

Il vous est maintenant possible de vous connecter avec l'utilisateur `pizza` / `pizzamdp`.

Pour changer le mot de passe, il vous faudra utiliser la commande suivante :

```
UPDATE mysql.user SET password=PASSWORD('nouveaumotdepasse') WHERE user="pizza";
```

ou **se connecter avec l'utilisateur pizza** puis :

```
SET password=PASSWORD('nouveaumotdepasse');
```

Attention

☢ Si vous ne changez pas d'utilisateur, vous réinitialisez le mot de passe de `root`

3.2.2 Utilisation d'un fichier sql

Il est possible de mettre l'ensemble des commandes SQL dans une fichier texte dont l'extension par principe sera `.sql`.
On exécutera alors l'ensemble du contenu de ce fichier en utilisant la commande suivante :

```
mysql -u root -p < monfichiersql.sql
```

ou

```
mysql -u root -p MaBaseDeDonnees < monfichiersql.sql
```

où :

- MaBaseDeDonnees correspond au nom de la base sur laquelle on va opérer les commandes
- monfichiersql.sql correspond au nom du fichier contenant les commandes sql

3.2.3 Connexion à une base de données

La base de données héberge plusieurs bases, il est donc nécessaire de spécifier celle sur laquelle on va travailler. Pour se faire, on peut l'indiquer à la connexion :

```
mysql -u pizza -p FirstPizza
mysql -u pizza -ppizzamdp FirstPizza
```

Remarque

✓ L'argument `-p` demande la saisie du mot de passe lors de la connexion à la base de données. Sinon, c'est celui de l'utilisateur connecté au système qui est utilisé. Il est possible de mettre le mot de passe après le `-p` mais dans ce cas votre historique de commandes trahira votre mot de passe

ou utiliser la commande :

```
mysql> use FirstPizza
```

3.2.4 Création des 2 tables de données

Nous allons maintenant créer les 2 tables dont les noms ont été précédemment définis.
Se déconnecter puis se connecter avec le login "pizza" :

```
mysql> exit;
eric@Tuxie:~$ mysql -u pizza -p FirstPizza
```

Puis créer les tables avec les commandes suivantes :

```
mysql> create table Pate (nom_pate VARCHAR (50));
mysql> create table Garniture (nom_garniture VARCHAR (50));
```

Perte de mémoire

Tester les 2 commandes suivantes et indiquez leurs rôles en le confirmant :

```
mysql> show databases;
mysql> select database ();
```

La base `information_schema` est la base native de mysql, il est donc possible de l'utiliser

```
mysql> show databases;
+--------------------+
| Database |
+--------------------+
| information_schema |
| FirstPizza |
| mysql |
| performance_schema |
| test |
+--------------------+
5 rows in set (0.01 sec)
mysql> use information_schema;
mysql> select database ();
+--------------------+
| database () |
+--------------------+
| information_schema |
+--------------------+
1 row in set (0.00 sec)
```

3.2.5 Garniture de la table

Les tables ainsi créées sont vides. Pour le constater, utiliser les commandes :

```
mysql> use FirstPizza;
mysql> show tables;
```

Nous allons maintenant remplir ces tables avec l'ensemble des éléments que nous avons trouvés. Pour ce faire plusieurs méthodes, manuelle, automatique ou par saisie au travers d'une IHM (Interface Homme Machine). Dans l'immédiat nous ne verrons que les deux premières.

Saisie Manuelle

À l'instar de ce que nous avons réalisé précédemment, il est possible d'ajouter des éléments dans notre table en commande SQL. Pour cela, la commande à utiliser est `INSERT`.

Attention

☢ Tout n'est pas indiqué, reprenez ce que vous avez déjà vu

```
mysql> insert into Pate set nom_pate="Classique";
```

Réaliser l'ajout de l'ensemble des pâtes proposées. Puis tester la présence de l'ensemble des données à l'aide de la commande : `select * from Pate;`

```
mysql> select * from Pate;
+-----------------+
| nom_pate |
+-----------------+
| Classique |
| Aux Herbes |
```

```
| Complete |
| Pomme de Terre |
| Au Lait |
| Farine de Mais |
+-----------------+
6 rows in set (0.00 sec)
```

Saisie Automatique

Comme nous l'avons vu la tâche d'insertion d'une donnée dans une table est répétitive. Nous allons donc utiliser un langage de programmation pour insérer les données.

— Python2 : Le module `MySQLdb`

— Python 3 : Le module `mysql.connector`

permettent de se connecter à une base de données et d'y effectuer des requêtes
En voici un exemple. [1]

Listing 3.4 – `ex_python_sql_2.py`

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
import MySQLdb as mdb
import sys

conn = mdb.connect( host="localhost", # votre hôte, normalement le serveur local
                user="pizza", # utilisateur
                passwd="pizzamdp", # mot de passe
                db="FirstPizza") # nom de la base de données

# Traitement des requêtes aussitôt
conn.autocommit(True)

# Il est nécessaire de créer un objet de type Curseur pour réaliser des requêtes
curs = conn.cursor()

try:
   curs.execute('insert into Garniture set nom_garniture="Tomates"')

except mdb.Error, e:
   print "Erreur %d: %s" % (e.args[0],e.args[1])
   sys.exit(1)

finally:
   curs.close()
   conn.close()
```

Listing 3.5 – `ex_python_sql_3.py`

```
#!/usr/bin/python3
# -*- coding: utf-8 -*-
import sys
import mysql.connector

conn = mysql.connector.connect(
```

1. Pour exécuter un programme Python :

```
chmod +x prog.py
./prog.py
```

```
                          user='pizza',
                          password='pizzamdp',
                          host='localhost',
                          database='FirstPizza'
)


# Traitement des requêtes aussitôt
conn.autocommit = True

# Il est nécessaire de créer un objet de type Curseur pour réaliser des requêtes
curs = conn.cursor()

try:
   curs.execute ('insert into Garniture set nom_garniture="Tomates"')

except mysql.connector.Error as err:
   print ("Erreur %d: %s" % (e.args[0],e.args[1]))
   sys.exit(1)

finally:
   curs.close()
   conn.close()
```

Remarque

✓ Oups je me suis trompé ...
dans ce cas, on vide la table et on recommence à l'aide de la commande `truncate table Garniture` ou on détruit la table par la commande `drop table Garniture`

Exercice

✓ Modifier cet exemple pour lire le fichier texte `garniture.txt` et insérer chacune des lignes dans la table Garniture. (Attention à ne pas ajouter 2 fois les tomates)

Listing 3.6 – `corr_garniture_2.py`

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
import MySQLdb as mdb
import sys

conn = mdb.connect( host="localhost", # votre hôte, normalement le serveur local
                user="pizza", # utilisateur
                passwd="pizzamdp", # mot de passe
                db="FirstPizza") # nom de la base de données

# Traitement des requêtes aussitôt
conn.autocommit(True)

# Il est nécessaire de créer un objet de type Curseur pour réaliser des requêtes
curs = conn.cursor()

# On ouvre le fichier "garniture.txt"
f = open('garniture.txt', 'r')

for line in f:
   requete = 'insert into Garniture set nom_garniture="' + line.rstrip() + '"'

   try:
      curs.execute(requete)

   except mdb.Error, e:
      print "Erreur %d: %s" % (e.args[0],e.args[1])
      sys.exit(1)

curs.close()
conn.close()

f.close ()
```

Listing 3.7 – `corr_garniture_3.py`

```
#!/usr/bin/python3
# -*- coding: utf-8 -*-
```

```
import sys
import mysql.connector

conn = mysql.connector.connect(
                          user='pizza',
                          password='pizzamdp',
                          host='localhost',
                          database='FirstPizza'
)

# Traitement des requêtes aussitôt
conn.autocommit = True

# Il est nécessaire de créer un objet de type Curseur pour réaliser des requêtes
curs = conn.cursor()

# On ouvre le fichier "garniture.txt"
f = open('garniture.txt', 'r')

for line in f:
    requete = 'insert into Garniture set nom_garniture="' + line.rstrip() + '"'

    try:
        curs.execute(requete)

    except mysql.connector.Error as err:
        print ("Erreur %d: %s" % (e.args[0],e.args[1]))
        sys.exit(1)

curs.close()
conn.close()

f.close ()
```

Exercice

✓ Vérifier que la table Garniture a bien été remplie

```
mysql> select * from Garniture;
```

3.3 On met le couvert

Nous disposons maintenant de 2 tables mais impossible de visualiser ces dernières sans rentrer dans le moteur de bases de données.

Nous allons donc voir l'affichage du contenu d'une table au travers d'un programme Python.

3.3.1 Affichage des données en Python

Voici un exemple de programme pour l'affichage de la table "Pate".

Listing 3.8 – `affichage_pate_2.py`

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
import MySQLdb as mdb
import sys

conn = mdb.connect( host="localhost", # votre hôte, normalement le serveur local
                user="pizza", # utilisateur
                passwd="pizzamdp", # mot de passe
                db="FirstPizza") # nom de la base de données

# Il est nécessaire de créer un objet de type Curseur pour réaliser des requêtes
curs = conn.cursor()

curs.execute("SELECT * FROM Pate")

rows = curs.fetchall()

for row in rows:
```

```
    print row[0]

curs.close()
conn.close()
```

Listing 3.9 – affichage_pate_3.py

```
#!/usr/bin/python3
# -*- coding: utf-8 -*-
import sys
import mysql.connector

conn = mysql.connector.connect(
                          user='pizza',
                          password='pizzamdp',
                          host='localhost',
                          database='FirstPizza'
)

# Traitement des requêtes aussitôt
conn.autocommit = True

# Il est nécessaire de créer un objet de type Curseur pour réaliser des requêtes
curs = conn.cursor()

curs.execute("SELECT * FROM Pate")

rows = curs.fetchall()

for row in rows:
    print (row[0])

curs.close()
conn.close()
```

Exercice

✓ Modifier ce programme pour afficher la table ”Garniture”

Listing 3.10 – corr_affichage_garniture_2.py

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
import MySQLdb as mdb
import sys

conn = mdb.connect( host="localhost", # votre hôte, normalement le serveur local
                user="pizza", # utilisateur
                passwd="pizzamdp", # mot de passe
                db="FirstPizza") # nom de la base de données

# Il est nécessaire de créer un objet de type Curseur pour réaliser des requêtes
curs = conn.cursor()

curs.execute("SELECT * FROM Garniture")

rows = curs.fetchall()

for row in rows:
    print row[0]

curs.close()
conn.close()
```

Listing 3.11 – corr_affichage_garniture_3.py

```
#!/usr/bin/python3
# -*- coding: utf-8 -*-
import sys
import mysql.connector

conn = mysql.connector.connect(
                          user='pizza',
                          password='pizzamdp',
```

```
                          host='localhost',
                          database='FirstPizza'
)

# Traitement des requêtes aussitôt
conn.autocommit = True

# Il est nécessaire de créer un objet de type Curseur pour réaliser des requêtes
curs = conn.cursor()

curs.execute("SELECT * FROM Garniture")

rows = curs.fetchall()

for row in rows:
   print (row[0])

curs.close()
conn.close()
```

L'option `ORDER BY` de MySQL permet de trier les données.

Exercice

✓ Rechercher son utilisation et appliquez-le aux 2 programmes précédents créés pour faire un tri alphabétique et inverse [1].

```
mysql> select * from Garniture ORDER BY nom_garniture;
mysql> select * from Pate ORDER BY nom_pate;
mysql> select * from Garniture ORDER BY nom_garniture DESC;
mysql> select * from Pate ORDER BY nom_pate DESC;
```

3.4 Hep garçon, s'il vous plait, auriez-vous ...?

L'un des buts recherché dans une base de données est de pouvoir réaliser des requêtes sur cette dernière et notamment de connaître l'existence ou non d'une donnée dans une table.
Pour se faire MySQL dispose de la commande `SELECT` qui s'exécute de la façon suivante :

```
SELECT champs FROM table WHERE condition;
```

Dans le cas d'une chaîne de caractère la condition peut être

- égale : `=`
- différent : `!=`

En voici un exemple.

```
mysql> select * from Garniture WHERE nom_garniture='Olives';
+---------------+
| nom_garniture |
+---------------+
| Olives |
+---------------+
1 row in set (0.00 sec)
```

Il est parfois difficile de trouver exactement le bon terme, aussi est-il possible de faire une recherche de type "RESSEMBLE À" ou "LIKE" en anglais. `LIKE` peut être utilisé avec des méta-caractères [2] tels que :

- `%` qui remplace n'importe quelle chaîne.
- _ qui remplace n'importe quel caractère.

En voici un exemple.

```
mysql> select * from Garniture WHERE nom_garniture LIKE 'O%';
+---------------+
| nom_garniture |
+---------------+
| Oignons |
```

1. Il vous est conseillé de réaliser votre commande SQL dans MySQL avant de l'appliquer aux programmes
2. Aussi appelés caractères joker, ces caractères remplacent d'autres caractères

```
| Olives |
| Oeuf |
| Orange |
+---------------+
4 rows in set (0.00 sec)
```

Exercice

✓ Écrire un programme qui demande une garniture à l'utilisateur et indique si cette garniture est disponible ou non.
Dans le cas où aucune garniture n'est trouvée, on cherchera si on dispose quelque chose d'approchant.
Exemple : "fromage" → "fromage de chèvre frais"

Remarque

✓ En Python2, la commande de saisie est `raw_input()`.En Python3, la commande saisie est `input()`.

```
Votre garniture désirée ? fromage
Garniture : fromage
Désolé, nous n'avons pas : fromage
Cependant nous pouvons vous proposer :
Fromage de chevre frais

Votre garniture désirée ? noisette
Garniture : noisette
Désolé, nous n'avons pas : noisette
Nous ne pouvons rien vous proposer qui pourrait satisfaire à votre demande.
```

Listing 3.12 – `corr_hep_garcon_2.py`

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
import MySQLdb as mdb
import sys

conn = mdb.connect( host="localhost", # votre hôte, normalement le serveur local
                user="pizza", # utilisateur
                passwd="pizzamdp", # mot de passe
                db="FirstPizza") # nom de la base de données

# Il est nécessaire de créer un objet de type Curseur pour réaliser des requêtes
curs = conn.cursor()

# Demande de la garniture
garniture = raw_input ("Votre garniture désirée ? ")
print ("Garniture : " + garniture)

# Rechercher de la garniture
curs.execute("SELECT * FROM Garniture WHERE nom_garniture=%s", [garniture])
rows = curs.fetchall()

# Nombre de résultats
nbre_resulats = len (rows)

if (nbre_resulats != 0 ):
   for row in rows:
      print row[0]
else:
   print ("Désolé, nous n'avons pas : " + (garniture))

   # Rechercher de la garniture
   curs.execute("SELECT * FROM Garniture WHERE nom_garniture LIKE %s", ["%" + garniture + "%"])
   rows = curs.fetchall()

   # Nombre de résultats
   nbre_resultats = len (rows)

   if (nbre_resultats != 0 ):
      print "Cependant nous pouvons vous proposer : "
      for row in rows:
         print row[0]
   else:
      print "Nous ne pouvons rien vous proposer qui pourrait satisfaire à votre demande."
```

```
curs.close()
conn.close()
```

Listing 3.13 – `corr_hep_garcon_3.py`

```
#!/usr/bin/python3
# -*- coding: utf-8 -*-
import sys
import mysql.connector

conn = mysql.connector.connect(
                           user='pizza',
                           password='pizzamdp',
                           host='localhost',
                           database='FirstPizza'
)

# Traitement des requêtes aussitôt
conn.autocommit = True

# Il est nécessaire de créer un objet de type Curseur pour réaliser des requêtes
curs = conn.cursor()

# Demande de la garniture
garniture = input ("Votre garniture désirée ? ")
print ("Garniture : " + garniture)

# Rechercher de la garniture
curs.execute("SELECT * FROM Garniture WHERE nom_garniture=%s", [garniture])
rows = curs.fetchall()

# Nombre de résultats
nbre_resulats = len (rows)

if (nbre_resulats != 0 ):
    for row in rows:
        print (row[0])
else:
    print ("Désolé, nous n'avons pas : " + (garniture))

    # Rechercher de la garniture
    curs.execute("SELECT * FROM Garniture WHERE nom_garniture LIKE %s", ["%" + garniture + "%"])
    rows = curs.fetchall()

    # Nombre de résultats
    nbre_resultats = len (rows)

    if (nbre_resultats != 0 ):
        print ("Cependant nous pouvons vous proposer : ")
        for row in rows:
            print (row[0])
    else:
        print ("Nous ne pouvons rien vous proposer qui pourrait satisfaire à votre demande.")

curs.close()
conn.close()
```

3.5 Garniture + Pâte = Pizza

Il est maintenant temps de créer une table de nos pizzas. Pour cela, plusieurs méthodes s'offrent à nous.

3.5.1 Une première Pizza un peu lourde

Il est possible de créer une table Pizza unique qui contiendra le type de pâte utilisé et les ingrédients. Le nombre d'ingrédients n'étant pas le même pour toutes les pizzas on fixera ce nombre et on ne remplira que ceux qui servent, les autres seront mis à `NULL`. `NULL` est un élément que l'on retrouve dans tous les langages de programmation que l'on peut associer à une valeur définie comme RIEN.
De manière à aller plus vite, on va donc définir nos ingrédients avec une valeur par défaut `NULL`.

Voici la table :

Pizza
nom_Pizza
nom_Pate
Garniture1
Garniture2
Garniture3
Garniture...
Garniture9

TABLE 3.1 : Table `Pizza`

Exercice

✓ Créer cette table MySQL et la remplir en composant 5 pizzas de votre choix. N'oubliez pas le NULL par défaut.

- l'option `DEFAULT NULL` permet d'indiquer que le champs est par défaut égal à NULL
- le séparateur , permet de définir les différents champs

Exemple

✓ `create table mapizza ( nom_pizza VARCHAR(50) not null, nom_pate VARCHAR(50) not null, ingredient varchar(50) default null );`

```
DROP TABLE IF EXISTS `Pizza`;

CREATE TABLE `Pizza` (
  `nom_Pizza` varchar(50) NOT NULL,
  `nom_Pate` varchar(50) NOT NULL,
  `Ingredient1` varchar(50) DEFAULT NULL,
  `Ingredient2` varchar(50) DEFAULT NULL,
  `Ingredient3` varchar(50) DEFAULT NULL,
  `Ingredient4` varchar(50) DEFAULT NULL,
  `Ingredient5` varchar(50) DEFAULT NULL,
  `Ingredient6` varchar(50) DEFAULT NULL,
  `Ingredient7` varchar(50) DEFAULT NULL,
  `Ingredient8` varchar(50) DEFAULT NULL,
  `Ingredient9` varchar(50) DEFAULT NULL
);

INSERT INTO `Pizza` SET nom_Pizza = "Marguerita", nom_Pate = "Classique", Ingredient1 = "Tomates", Ingredient2 = "
     Mozarella";

INSERT INTO `Pizza` SET nom_Pizza = "Asperges et Jambon", nom_Pate = "Complete", Ingredient1 = "Asperges", Ingredient2
      = "Mozarella", Ingredient3 = "Jambon";

INSERT INTO `Pizza` SET nom_Pizza = "Hawaienne", nom_Pate = "Classique", Ingredient1 = "Jambon", Ingredient2 = "Tomate
     ", Ingredient3 = "Gruyere", Ingredient4 = "Ananas";

INSERT INTO `Pizza` SET nom_Pizza = "Napolitaine", nom_Pate = "Classique", Ingredient1 = "Mozzarella", Ingredient2 = "
     Anchois", Ingredient3 = "Tomates";

INSERT INTO `Pizza` SET nom_Pizza = "Crevettes Paprika", nom_Pate = "Aux herbes", Ingredient1 = "Gruyere", Ingredient2
      = "Crevettes";
```

Réfléchissons

- Quelles sont les conséquences de cette structure sur l'espace disque ?
- Que devrez vous faire si vous avez une pizza avec 10 éléments ?
- Quel est l'intérêt d'avoir créé deux autres tables ?
- Recherchez les pizzas avec un ingrédient que vous avez utilisé dans l'une de vos pizzas puis avec un ingrédient que vous n'avez pas utilisé.

Exercice

✓ Écrire l'algorithme de recherche d'une pizza avec un ingrédient défini...

```
Pour chaque pizza dans la table pizza,
   Pour chaque ingrédient de cette pizza non NULL,
      Comparer l'ingrédient avec celui recherché
         Si comparaison ok - Ingrédient trouvé - sortir
Ingrédient non trouvé
```

En fait nous ne pouvons pas utiliser le moteur de bases de données car les ingrédients sont renseignés de manière multiples.

3.5.2 Une Pizza allégée

Comme nous l'avons constaté précédemment, nous n'avons pas utilisé les tables "Garniture" et "Pate" dans notre table Pizza. Nous allons corriger cette problématique mais afin

- D'éviter les erreurs (faute de frappe dans la saisie d'un ingrédient ou d'une pâte),
- De permettre la modification des appellations (exemple : fromage de chèvre frais → fromage de chèvre),
- D'économiser de l'espace disque (nous prenons 50 caractères par ingrédient renseigné ou non),

nous allons utiliser des identifiants pour chacun des éléments.

Pour se faire, ajouter un champ

- `id_Garniture` à la table Garniture
- `id_Pate` à la table Pate
- `id_Pizza` à la table Pizza

. Nous déclarerons ce champ en `AUTO_INCREMENT` de manière à ce que le numéro d'identifiant soit renseigné automatiquement. Il serait bien sûr possible de saisir ces identifiants manuellement mais soyons fainéants, n'est ce pas là la qualité première d'un informaticien ?

L'auto-incrément nous oblige à indiquer que cet élément est une clé primaire. Ceci signifie que **la valeur de cet identifiant sera unique pour la table**.

Le mot clé `FIRST` nous permet de disposer des identifiants en tant que premier élément de la table (ce qui humainement parlant paraît plus logique).

```
ALTER TABLE `Pate` ADD `id_Pate` INT NOT NULL AUTO_INCREMENT FIRST, ADD PRIMARY KEY ( `id_Pate` );

ALTER TABLE `Garniture` ADD `id_Garniture` INT NOT NULL AUTO_INCREMENT FIRST, ADD PRIMARY KEY ( `id_Garniture` );

ALTER TABLE `Pizza` ADD `id_Pizza` INT NOT NULL AUTO_INCREMENT FIRST, ADD PRIMARY KEY ( `id_Pizza` );
```

Exercice

✓ Consulter les nouvelles tables ainsi créées (`SELECT`)

```
SELECT * FROM `Pate`;

SELECT * FROM `Garniture`;

SELECT * FROM `Pizza`;
```

Nous allons modifier la table Pizza pour prendre en compte les identifiants uniques. Une problématique apparaît, les identifiants sont des nombres, les champs que nous avons créés sont des chaînes de caractères. Nous allons donc détruire notre table pour la reconstruire.
La destruction d'une table peut être réalisée par la commande `DROP TABLE`.

Exercice

✓ Créer et remplir la `nouvelle` table Pizza.
Nous nommerons les champs de manière préfixée par fk (foreign key), pour indiquer un lien vers une autre table.
Un identifiant en mode auto-incrément commençant à 1, nous initialiserons les valeurs à 0 par défaut. Il est possible de conserver le `NULL` mais prendre 0 permet de conserver une cohérence dans le typage de la donnée.
Au lieu de la garniture "Tomates" nous aurons donc par exemple `fk_garniture1 = 3`

Colonne	Type	Valeur par défaut
idPizza	INT	Autoincrement
nomPizza	VARCHAR (50)	Aucune
fk_id_Pate	INT	0
fk_id_Garniture1	INT	0
fk_id_Garniture2	INT	0
fk_id_Garniture3	INT	0
fk_id_Garniture...	INT	0
fk_id_Garniture9	INT	0

TABLE 3.2 : Caractéristiques de la table `Pizza`

Remarque

✓ Afin de pouvoir indiquer un élément de type AUTO_INCREMENT, il est nécessaire de déclarer ce dernier en tant que clé primaire. Il faut donc ajouter lors de la création de la table la ligne suivante : `PRIMARY KEY (id_pizza)`.

```
DROP TABLE IF EXISTS `Pizza`;

CREATE TABLE `Pizza` (
  `id_Pizza` INT NOT NULL AUTO_INCREMENT,
  `nom_Pizza` varchar(50) NOT NULL,
  `fk_id_Pate` INT DEFAULT 0,
  `fk_id_Garniture1` INT DEFAULT 0,
  `fk_id_Garniture2` INT DEFAULT 0,
  `fk_id_Garniture3` INT DEFAULT 0,
  `fk_id_Garniture4` INT DEFAULT 0,
  `fk_id_Garniture5` INT DEFAULT 0,
  `fk_id_Garniture6` INT DEFAULT 0,
  `fk_id_Garniture7` INT DEFAULT 0,
  `fk_id_Garniture8` INT DEFAULT 0,
  `fk_id_Garniture9` INT DEFAULT 0,
  PRIMARY KEY (`id_Pizza`)
);

INSERT INTO `Pizza` SET nom_Pizza = "Marguerita", fk_id_Pate = 1, fk_id_Garniture1 = 1, fk_id_Garniture2 = 10;

INSERT INTO `Pizza` SET nom_Pizza = "Asperges et Jambon", fk_id_Pate = 3, fk_id_Garniture1 = 27, fk_id_Garniture2 =
     10, fk_id_Garniture3= 26;

INSERT INTO `Pizza` SET nom_Pizza = "Hawaienne", fk_id_Pate = 1, fk_id_Garniture1 = 26, fk_id_Garniture2 = 1,
     fk_id_Garniture3 = 3, fk_id_Garniture4 = 23;

INSERT INTO `Pizza` SET nom_Pizza = "Napolitaine", fk_id_Pate = 1, fk_id_Garniture1 = 10, fk_id_Garniture2 = 30,
     fk_id_Garniture3 = 1;

INSERT INTO `Pizza` SET nom_Pizza = "Crevettes Paprika", fk_id_Pate = 2, fk_id_Garniture1 = 3, fk_id_Garniture2 = 34;
```

Réfléchissons

- Quelles sont les conséquences de cette structure sur l'espace disque ?
- Que devrez vous faire si vous avez une pizza avec 11 éléments ?
- Recherchez les pizzas avec un ingrédient que vous avez utilisé dans l'une de vos pizzas puis avec un ingrédient que vous n'avez pas utilisé.

Exercice

✓ Écrire l'algorithme de recherche d'un ingrédient ...

```
Saisie de l'ingrédient recherché
Rechercher dans la table Garniture si l'élément existe
Si l'ingrédient existe
   stocker l'identifiant de ce dernier
```

```
Sinon
   sortir

Pour chaque pizza dans la table pizza,
   Pour chaque ingrédient de cette pizza différent de 0
      Comparer l'identifiant avec l'identifiant trouvé précédemment
         Si comparaison ok - Ingrédient trouvé - sortir

Ingrédient non trouvé
```

L'algorithme se complexifie.

Exercice

✓ Écrire un programme qui affiche toutes vos pizzas.

```
Nom de la pizza : Napolitaine
Nom de la pâte : Classique
Garniture : Mozzarella
Garniture : Anchois
Garniture : Tomates
```

Listing 3.14 – `corr_affichage_pizza_2.py`

```python
#!/usr/bin/python
# -*- coding: utf-8 -*-
import MySQLdb as mdb
import sys

conn = mdb.connect( host="localhost", # votre hôte, normalement le serveur local
                user="pizza", # utilisateur
                passwd="pizzamdp", # mot de passe
                db="FirstPizza") # nom de la base de données

# Il est nécessaire de créer un objet de type Curseur pour réaliser des requêtes
curs = conn.cursor()

curs.execute("SELECT * FROM Pizza")

rows = curs.fetchall()

for row in rows:
   print ("Nom de la pizza : " + row[1])

   # On recherche l identifiant associé à la pâte
   curs.execute("SELECT nom_Pate FROM Pate WHERE id_Pate = %d" % (row[0]))
   pate = curs.fetchone()

   if pate is not None:
      print("Nom de la pâte : %s" % (pate))
   else:
      print ("Pâte introuvable - cohésion de la base de données incorrecte")

   for col in range(2,11):

      if row[col] != 0 :
         curs.execute("SELECT nom_garniture FROM Garniture where id_Garniture=%d" % (row[col]))

         nom_garniture = curs.fetchone()
         if nom_garniture is not None:
            print("Garniture : %s" % nom_garniture)
         else:
            print ("Ingrédient introuvable - cohésion de la base de données incorrecte")

   # Pour séparer les différentes pizzas
   print

curs.close()
conn.close()
```

Listing 3.15 – corr_affichage_pizza_3.py

```
#!/usr/bin/python3
# -*- coding: utf-8 -*-
import sys
import mysql.connector

conn = mysql.connector.connect(
                           user='pizza',
                           password='pizzamdp',
                           host='localhost',
                           database='FirstPizza'
)


# Traitement des requêtes aussitôt
conn.autocommit = True

# Il est nécessaire de créer un objet de type Curseur pour réaliser des requêtes
curs = conn.cursor()

curs.execute("SELECT * FROM Pizza")

rows = curs.fetchall()

for row in rows:
   print ("Nom de la pizza : " + row[1])

   # On recherche l identifiant associé à la pâte
   curs.execute("SELECT nom_Pate FROM Pate WHERE id_Pate = %d" % (row[0]))
   pate = curs.fetchone()

   if pate is not None:
      print("Nom de la pâte : %s" % (pate))
   else:
      print ("Pâte introuvable - cohésion de la base de données incorrecte")

   for col in range(2,11):

      if row[col] != 0 :
         curs.execute("SELECT nom_garniture FROM Garniture where id_Garniture=%d" % (row[col]))

         nom_garniture = curs.fetchone()
         if nom_garniture is not None:
            print("Garniture : %s" % nom_garniture)
         else:
            print ("Ingrédient introuvable - cohésion de la base de données incorrecte")

   # Pour séparer les différentes pizzas
   print ()

curs.close()
conn.close()
```

3.5.3 Une Pizza poids forme

Notre table est maintenant optimisée concernant l'espace disque, reste les problèmes suivants :

— le nombre d'ingrédients est limité à 9 or la plupart du temps, cet ensemble n'est pas utilisé (une marghérita prendra 2 éléments de garniture : Tomates et Mozzarella, donc 7 espaces vides.

— la recherche d'un ingrédient est difficile

Exercice

✓ Réfléchir à la construction d'une table qui associerait les ingrédients à la pizza. Construire cette table. Y insérer les données.

```
DROP TABLE IF EXISTS `GarniturePizza`;

CREATE TABLE `GarniturePizza` (
  `fk_id_Pizza` INT NOT NULL,
  `fk_id_Garniture` INT NOT NULL
```

```
);


INSERT INTO `GarniturePizza` SET fk_id_Pizza = 1, fk_id_Garniture = 1;
INSERT INTO `GarniturePizza` SET fk_id_Pizza = 1, fk_id_Garniture = 10;

INSERT INTO `GarniturePizza` SET fk_id_Pizza = 2, fk_id_Garniture = 27;
INSERT INTO `GarniturePizza` SET fk_id_Pizza = 2, fk_id_Garniture = 10;
INSERT INTO `GarniturePizza` SET fk_id_Pizza = 2, fk_id_Garniture = 26;

INSERT INTO `GarniturePizza` SET fk_id_Pizza = 3, fk_id_Garniture = 26;
INSERT INTO `GarniturePizza` SET fk_id_Pizza = 3, fk_id_Garniture = 1;
INSERT INTO `GarniturePizza` SET fk_id_Pizza = 3, fk_id_Garniture = 3;
INSERT INTO `GarniturePizza` SET fk_id_Pizza = 3, fk_id_Garniture = 23;

INSERT INTO `GarniturePizza` SET fk_id_Pizza = 4, fk_id_Garniture = 10;
INSERT INTO `GarniturePizza` SET fk_id_Pizza = 4, fk_id_Garniture = 30;
INSERT INTO `GarniturePizza` SET fk_id_Pizza = 4, fk_id_Garniture = 1;

INSERT INTO `GarniturePizza` SET fk_id_Pizza = 5, fk_id_Garniture = 3;
INSERT INTO `GarniturePizza` SET fk_id_Pizza = 5, fk_id_Garniture = 34;
```

Exercice

✓ Détruire les éléments rendus inutiles dans la table `pizza`

```
ALTER TABLE `Pizza` DROP `fk_id_Garniture1`;
ALTER TABLE `Pizza` DROP `fk_id_Garniture2`;
ALTER TABLE `Pizza` DROP `fk_id_Garniture3`;
ALTER TABLE `Pizza` DROP `fk_id_Garniture4`;
ALTER TABLE `Pizza` DROP `fk_id_Garniture5`;
ALTER TABLE `Pizza` DROP `fk_id_Garniture6`;
ALTER TABLE `Pizza` DROP `fk_id_Garniture7`;
ALTER TABLE `Pizza` DROP `fk_id_Garniture8`;
ALTER TABLE `Pizza` DROP `fk_id_Garniture9`;
```

Exercice

✓ Écrire un programme qui affiche tous les ingrédients utilisés dans **vos** pizzas. Utilisez l'option `DISTINCT` pour ne pas avoir d'éléments en double

Listing 3.16 – `corr_forme_garniture_2.py`

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
import MySQLdb as mdb
import sys

conn = mdb.connect( host="localhost", # votre hôte, normalement le serveur local
                user="pizza", # utilisateur
                passwd="pizzamdp", # mot de passe
                db="FirstPizza") # nom de la base de données

# Il est nécessaire de créer un objet de type Curseur pour réaliser des requêtes
curs = conn.cursor()

curs.execute("SELECT DISTINCT fk_id_Garniture FROM GarniturePizza")

rows = curs.fetchall()

for row in rows:

   # On recherche l identifiant associé à la garniture
   curs.execute("SELECT nom_Garniture FROM Garniture WHERE id_Garniture = %d" % (row[0]))
   garniture = curs.fetchone()

   if garniture is not None:
      print("Nom de l'ingrédient : %s" % (garniture))
   else:
      print ("Garniture introuvable - cohésion de la base de données incorrecte")

   # Pour séparer les différentes garnitures
```

```
    print

curs.close()
conn.close()
```

Listing 3.17 – corr_forme_garniture_3.py

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
import MySQLdb as mdb
import sys

conn = mdb.connect( host="localhost", # votre hôte, normalement le serveur local
                user="pizza", # utilisateur
                passwd="pizzamdp", # mot de passe
                db="FirstPizza") # nom de la base de données

# Il est nécessaire de créer un objet de type Curseur pour réaliser des requêtes
curs = conn.cursor()

curs.execute("SELECT DISTINCT fk_id_Garniture FROM GarniturePizza")

rows = curs.fetchall()

for row in rows:

   # On recherche l identifiant associé à la garniture
   curs.execute("SELECT nom_Garniture FROM Garniture WHERE id_Garniture = %d" % (row[0]))
   garniture = curs.fetchone()

   if garniture is not None:
      print("Nom de l'ingrédient : %s" % (garniture))
   else:
      print ("Garniture introuvable - cohésion de la base de données incorrecte")

   # Pour séparer les différentes garnitures
   print

curs.close()
conn.close()
```

Exercice

✓ Écrire un programme qui affiche toutes les pizzas que vous proposez avec leur composition.

Listing 3.18 – corr_forme_affichage_pizza_2.py

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
import MySQLdb as mdb
import sys

conn = mdb.connect( host="localhost", # votre hôte, normalement le serveur local
                user="pizza", # utilisateur
                passwd="pizzamdp", # mot de passe
                db="FirstPizza") # nom de la base de données

# Il est nécessaire de créer un objet de type Curseur pour réaliser des requêtes
curs = conn.cursor()

curs.execute("SELECT * FROM Pizza")

rows = curs.fetchall()

for row in rows:
   print ("Nom de la pizza : " + row[1])

   #~ On recherche l identifiant associé à la pâte

   curs.execute("SELECT nom_Pate FROM Pate WHERE id_Pate = %d" % (row[2]))
   nom_pate = curs.fetchone()

   if nom_pate is not None:
```

```
        print("Nom de la pâte : %s" % (nom_pate))
    else:
        print ("Pâte introuvable - cohésion de la base de données incorrecte")

    #~ Rechercher l ensemble des clés avec l identifiant de la pizza
    curs.execute ("SELECT fk_id_Garniture FROM GarniturePizza WHERE fk_id_Pizza = %d" % row[0])

    tab_garnitures = curs.fetchall()

    for fk_garniture in tab_garnitures:

        # On recherche l identifiant associé à la garniture
        curs.execute("SELECT nom_Garniture FROM Garniture WHERE id_Garniture = %d" % fk_garniture)
        garniture = curs.fetchone()

        if garniture is not None:
            print("Nom de l'ingrédient : %s" % (garniture))
        else:
            print ("Garniture introuvable - cohésion de la base de données incorrecte")

    # Pour séparer les différentes pizzas
    print

curs.close()
conn.close()
```

Listing 3.19 – `corr_forme_affichage_pizza_3.py`

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
import MySQLdb as mdb
import sys

conn = mdb.connect( host="localhost", # votre hôte, normalement le serveur local
                user="pizza", # utilisateur
                passwd="pizzamdp", # mot de passe
                db="FirstPizza") # nom de la base de données

# Il est nécessaire de créer un objet de type Curseur pour réaliser des requêtes
curs = conn.cursor()

curs.execute("SELECT * FROM Pizza")

rows = curs.fetchall()

for row in rows:
    print ("Nom de la pizza : " + row[1])

    #~ On recherche l identifiant associé à la pâte

    curs.execute("SELECT nom_Pate FROM Pate WHERE id_Pate = %d" % (row[2]))
    nom_pate = curs.fetchone()

    if nom_pate is not None:
        print("Nom de la pâte : %s" % (nom_pate))
    else:
        print ("Pâte introuvable - cohésion de la base de données incorrecte")

    #~ Rechercher l ensemble des clés avec l identifiant de la pizza
    curs.execute ("SELECT fk_id_Garniture FROM GarniturePizza WHERE fk_id_Pizza = %d" % row[0])

    tab_garnitures = curs.fetchall()

    for fk_garniture in tab_garnitures:

        # On recherche l identifiant associé à la garniture
        curs.execute("SELECT nom_Garniture FROM Garniture WHERE id_Garniture = %d" % fk_garniture)
        garniture = curs.fetchone()

        if garniture is not None:
            print("Nom de l'ingrédient : %s" % (garniture))
        else:
            print ("Garniture introuvable - cohésion de la base de données incorrecte")
```

```
    # Pour séparer les différentes pizzas
    print

curs.close()
conn.close()
```

Réfléchissons

— Quelles sont les conséquences de cette structure sur l'espace disque ?
— Que devrez vous faire si vous avez une pizza avec 11 éléments ?

3.6 Bilan

Une règle fondamentale en base de données

Dans une base de données, une information ne doit pas apparaître de multiples fois si elle n'est pas différente.

Exemple

✓ Les pizzas `Marguerita` et `Reine` sont fabriquées avec une pâte à pain.
L'information "Pâte à pain" ne doit pas apparaître en tant que telle dans vos recettes. Seule une **référence** à ce type de pâte doit exister.
En numérotant la "Pâte à pain" comme étant l'entrée numéro `n` de la table "Pates", mes deux recettes précédentes peuvent utiliser l'entrée "3" de la table "Pates" au lieu de l'information "Pâte à pain".

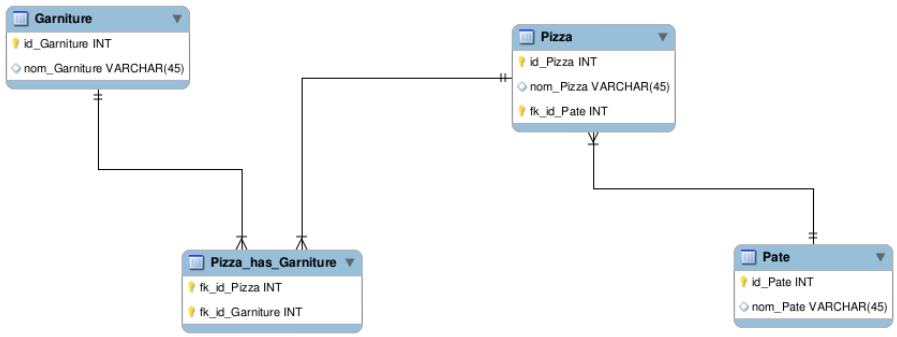


FIGURE 3.1 – Schéma de la base de données Pizza

Liaison 1 ↔ 1

Une liaison 1 pour 1 n'a normalement pas lieu d'exister, en effet, l'information peut être stockée dans la table elle même. Cependant cette dernière est notamment utilisée pour éviter les erreurs de saisie.

Exemple

✓ Le prix d'une pizza s'il est toujours différent n'a aucun intérêt à figurer sur une table séparée. Par contre, si ce prix est le même pour plusieurs pizzas, il est intéressant de le mettre de manière séparée.

Liaison 1 ↔ n

Une liaison 1 pour n signifie qu'un élément est composé d'un autre élément.

Exemple

✓ Il existe n pizzas qui sont composées à l'aide d'**UNE** pâte à pain.

Liaison n ↔ n

Une liaison n pour n signifie que **des** éléments sont composés de **plusieurs** autres éléments.

Exemple

✓ La `Sauce Tomate` est utilisée dans `plusieurs` pizzas.
Une pizza est composée de plusieurs éléments dont le nombre est variable.

3.7 Sauvegarde

Retrouver la base de données telle que vous devriez l'obtenir sur le fichier :

`Bases/FirstPizza/FirstPizza1.sql`

Chapitre 4

Pizzaïolo

4.1 Vite

Les clients souhaitent connaître le temps de préparation d'une pizza, ce temps n'impacte pas le prix mais le délai de livraison. Rajouter le champs `tps_cuisson` dans la table pizza. Compléter ce champs en vous inspirant des exemples donnés ci-dessous.

Pizza	Temps de Cuisson
Marguerita	20 min.
Asperges et jambon	10 min.
Hawaienne	20 min.
Napolitaine	15 min.
Crevettes Paprika	25 min. [a]

Table 4.1 : Temps de cuisson

a. 10 minutes dans le four, travail hors du four, 10 minutes de nouveau dans le four.

Pour ajouter le champs `temps_cuisson` dans la table `Pizza`, on utilisera la commande suivante :

Listing 4.1 – `corr_forme_affichage_pizza_3.py`

```
ALTER TABLE `Pizza` ADD `tps_cuisson` TIME NOT NULL
```

Pour mettre à jour le champs avec sa valeur, on utilisera une commande de type :

Listing 4.2 – `corr_forme_affichage_pizza_3.py`

```
UPDATE TABLE `Pizza` SET ... WHERE ...
```

Le format de données `Time` est de la forme : `HH:MM:SS`

```
update Pizza set tps_cuisson="00:10:00" WHERE nom_Pizza="Asperges et Jambon";
update Pizza set tps_cuisson="00:20:00" WHERE nom_Pizza="Marguerita";
update Pizza set tps_cuisson="00:20:00" WHERE nom_Pizza="Hawaienne";
update Pizza set tps_cuisson="00:15:00" WHERE nom_Pizza="Napolitaine";
update Pizza set tps_cuisson="00:25:00" WHERE nom_Pizza="Crevettes Paprika";
```

4.2 Un collègue bien impatient ...

Votre collègue vous indique qu'il a ajouté le champs `temps_repos` dans la table `Pate` par la commande suivante :

```
ALTER TABLE `Pate` ADD `tps_repos` TIME NOT NULL
```

Compléter les temps de repos en vous inspirant de la table suivante [1] :

1. En fait le temps de repos d'une pâte dépend **surtout** de la température à laquelle elle est exposée

Pate	Temps de Repos
Classique	1h30
Aux herbes	1h00
Complete	2h00
Pomme de terre	0h35
Au lait	0h55
Farine de Mais	0h25

TABLE 4.2 : Temps de repos

```
update Pate set tps_repos="01:30:00" WHERE id_Pate=1;
update Pate set tps_repos="01:00:00" WHERE id_Pate=2;
update Pate set tps_repos="02:00:00" WHERE id_Pate=3;
update Pate set tps_repos="00:35:00" WHERE id_Pate=4;
update Pate set tps_repos="00:55:00" WHERE id_Pate=5;
update Pate set tps_repos="00:25:00" WHERE id_Pate=6;
```

Votre collègue a remarqué que depuis cette modification, le programme d'affichage des pizzas ne fonctionne plus.

Exercice

✓ Trouvez la raison pour laquelle votre programme ne fonctionne plus. **Il ne vous est pas demandé de le corriger**

Nos programmes ont utilisé jusqu'alors des références à des positions relatives (0,1,2...), l'option FIRST a décalé l'ensemble de ces références. Programmer avec des positions relatives n'est pas forcément une bonne méthode voir c'est une mauvaise méthode.

4.2.1 Dictionnaire de données

Afin de pallier à cette problématique, nous allons utiliser un dictionnaire de données plutôt qu'un tableau. Cette fonctionnalité est possible par l'utilisation de l'élément `cursors` de la librairie `MySQLdb`.

Cette utilisation doit s'accompagner de la spécitification `conn.cursor(dictionary=True)` dans le cadre de l'utilisation de Python3.

Voici un exemple de code :

Listing 4.3 – `dict_pizzas_2.py`

```
#!/usr/bin/python
# -*- coding: utf-8 -*-

import MySQLdb
import MySQLdb.cursors
import datetime

database = MySQLdb.connect(host = "localhost", user = "pizza", passwd = "pizzamdp", db = "FirstPizza", cursorclass=
     MySQLdb.cursors.DictCursor)
curs = database.cursor()

curs.execute("SELECT * FROM Pizza")
pizzas = curs.fetchall()

for pizza in pizzas:
   print ("Nom : " + pizza ['nom_Pizza'])
   print ("Temps de cuisson : " + str(pizza['tps_cuisson']))

   curs2 = database.cursor()

   curs2.execute("SELECT * FROM Pate WHERE id_Pate = %d" % pizza['fk_id_Pate'])
```

```
    # Il ne peut y avoir qu'une pâte
    pate= curs2.fetchone()

    print ("Nom de la pate : " + pate['nom_pate'])
    print ("Temps de repos de la pate : " + str(pate['tps_repos']))

    print
```

Listing 4.4 – `dict_pizzas_3.py`

```
#!/usr/bin/python3
# -*- coding: utf-8 -*-
import sys
import mysql.connector
import datetime

conn = mysql.connector.connect(
                            user='pizza',
                            password='pizzamdp',
                            host='localhost',
                            database='FirstPizza'
)


# Traitement des requêtes aussitôt
conn.autocommit = True

# Il est nécessaire de créer un objet de type Curseur pour réaliser des requêtes
curs = conn.cursor(dictionary=True)

curs.execute("SELECT * FROM Pizza")
pizzas = curs.fetchall()

for pizza in pizzas:
    print ("Nom : " + pizza ['nom_Pizza'])
    print ("Temps de cuisson : " + str(pizza['tps_cuisson']))

    curs2 = conn.cursor(dictionary=True)

    curs2.execute("SELECT * FROM Pate WHERE id_Pate = %d" % pizza['fk_id_Pate'])

    # Il ne peut y avoir qu'une pâte
    pate= curs2.fetchone()

    print ("Nom de la pate : " + pate['nom_pate'])
    print ("Temps de repos de la pate : " + str(pate['tps_repos']))

    print ()
```

Exercice

✓ Compléter le programme avec le reste des informations disponibles sur les Pizzas.

Listing 4.5 – `dict_all_pizzas_2.py`

```
#!/usr/bin/python
# -*- coding: utf-8 -*-

import MySQLdb
import MySQLdb.cursors
import datetime

database = MySQLdb.connect(host = "localhost", user = "pizza", passwd = "pizzamdp", db = "FirstPizza", cursorclass=
      MySQLdb.cursors.DictCursor)
c = database.cursor()

c.execute("SELECT * FROM Pizza")
pizzas = c.fetchall()

for pizza in pizzas:
    print ("Nom : " + pizza ['nom_Pizza'])
    print ("Temps de cuisson : " + str(pizza['tps_cuisson']))
```

```
    # Détermination de la pâte

    c2 = database.cursor()
    c2.execute("SELECT * FROM Pate WHERE id_Pate = %d" % pizza['fk_id_Pate'])

    # Il ne peut y avoir qu'une pâte
    pate= c2.fetchone()

    print ("Nom de la pate : " + pate['nom_pate'])
    print ("Temps de repos de la pate : " + str(pate['tps_repos']))

    # Détermination des garnitures
    c2.execute("SELECT fk_id_Garniture FROM GarniturePizza WHERE fk_id_Pizza = %d" % pizza['id_Pizza'])
    garnitures = c2.fetchall()

    for garniture in garnitures:
        c3 = database.cursor()
        c3.execute("SELECT nom_garniture FROM Garniture WHERE id_Garniture = %d" % garniture['fk_id_Garniture'])

        # Il ne peut y avoir qu'une garniture par id
        nom_garniture = c3.fetchone()

        print ("Garniture : " + nom_garniture['nom_garniture'])

        c3.close

    print

    c2.close

c.close
```

Listing 4.6 – `dict_all_pizzas_3.py`

```
#!/usr/bin/python3
# -*- coding: utf-8 -*-
import sys
import mysql.connector
import datetime

database = mysql.connector.connect(
                            user='pizza',
                            password='pizzamdp',
                            host='localhost',
                            database='FirstPizza'
)


# Traitement des requêtes aussitôt
database.autocommit = True

# Il est nécessaire de créer un objet de type Curseur pour réaliser des requêtes
curs = database.cursor(dictionary=True)

curs.execute("SELECT * FROM Pizza")
pizzas = curs.fetchall()

for pizza in pizzas:
    print ("Nom : " + pizza ['nom_Pizza'])
    print ("Temps de cuisson : " + str(pizza['tps_cuisson']))

    # Détermination de la pâte

    curs2 = database.cursor(dictionary=True)
    curs2.execute("SELECT * FROM Pate WHERE id_Pate = %d" % pizza['fk_id_Pate'])

    # Il ne peut y avoir qu'une pâte
    pate= curs2.fetchone()

    print ("Nom de la pate : " + pate['nom_pate'])
    print ("Temps de repos de la pate : " + str(pate['tps_repos']))

    # Détermination des garnitures
    curs2.execute("SELECT fk_id_Garniture FROM GarniturePizza WHERE fk_id_Pizza = %d" % pizza['id_Pizza'])
```

```
    garnitures = curs2.fetchall()

    for garniture in garnitures:
        curs3 = database.cursor(dictionary=True)
        curs3.execute("SELECT nom_garniture FROM Garniture WHERE id_Garniture = %d" % garniture['fk_id_Garniture'])

        # Il ne peut y avoir qu'une garniture par id
        nom_garniture = curs3.fetchone()

        print ("Garniture : " + nom_garniture['nom_garniture'])

        curs3.close

    print ()

    curs2.close

curs.close
```

Exercice

✓ Simplifier votre programme en utilisant le mot clé SQL `IN` qui permet de sélectionner l'ensemble des éléments qui sont dans un ensemble défini. On utilisera la forme de IN suivante : `IN (SELECT ...)`

Listing 4.7 – `dict_all_in_pizzas_2.py`

```
#!/usr/bin/python
# -*- coding: utf-8 -*-

import MySQLdb
import MySQLdb.cursors
import datetime

database = MySQLdb.connect(host = "localhost", user = "pizza", passwd = "pizzamdp", db = "FirstPizza", cursorclass=
      MySQLdb.cursors.DictCursor)
c = database.cursor()

c.execute("SELECT * FROM Pizza")
pizzas = c.fetchall()

for pizza in pizzas:
    print ("Nom : " + pizza ['nom_Pizza'])
    print ("Temps de cuisson : " + str(pizza['tps_cuisson']))

    # Détermination de la pâte

    c2 = database.cursor()
    c2.execute("SELECT * FROM Pate WHERE id_Pate = %d" % pizza['fk_id_Pate'])

    # Il ne peut y avoir qu'une pâte
    pate= c2.fetchone()

    print ("Nom de la pate : " + pate['nom_pate'])
    print ("Temps de repos de la pate : " + str(pate['tps_repos']))

    # Détermination des garnitures
    c2.execute("SELECT nom_garniture FROM Garniture WHERE id_Garniture IN (SELECT fk_id_Garniture FROM GarniturePizza
          WHERE fk_id_Pizza = %d)" % pizza['id_Pizza'])

    garnitures = c2.fetchall()

    for garniture in garnitures:
        print ("Garniture : " + garniture['nom_garniture'])
    print

    c2.close

c.close
```

Listing 4.8 – `dict_all_in_pizzas_3.py`

```
#!/usr/bin/python3
```

```
# -*- coding: utf-8 -*-
import sys
import mysql.connector
import datetime

database = mysql.connector.connect(
                           user='pizza',
                           password='pizzamdp',
                           host='localhost',
                           database='FirstPizza'
)


# Traitement des requêtes aussitôt
database.autocommit = True

# Il est nécessaire de créer un objet de type Curseur pour réaliser des requêtes
curs = database.cursor(dictionary=True)

curs.execute("SELECT * FROM Pizza")
pizzas = curs.fetchall()

for pizza in pizzas:
   print ("Nom : " + pizza ['nom_Pizza'])
   print ("Temps de cuisson : " + str(pizza['tps_cuisson']))

   # Détermination de la pâte

   c2 = database.cursor(dictionary=True)
   c2.execute("SELECT * FROM Pate WHERE id_Pate = %d" % pizza['fk_id_Pate'])

   # Il ne peut y avoir qu'une pâte
   pate= c2.fetchone()

   print ("Nom de la pate : " + pate['nom_pate'])
   print ("Temps de repos de la pate : " + str(pate['tps_repos']))

   # Détermination des garnitures
   c2.execute("SELECT nom_garniture FROM Garniture WHERE id_Garniture IN (SELECT fk_id_Garniture FROM GarniturePizza
         WHERE fk_id_Pizza = %d)" % pizza['id_Pizza'])

   garnitures = c2.fetchall()

   for garniture in garnitures:
      print ("Garniture : " + garniture['nom_garniture'])

   print ()

   c2.close


curs.close
```

4.3 Sauvegarde

Retrouver la base de données telle que vous devriez l'obtenir sur le fichier :

`Bases/FirstPizza/FirstPizza2.sql`

Chapitre 5

Devoir à la maison

Durée estimée pour la réalisation de ce devoir : 3h00

5.1 Intitulé du devoir

Après le succès de l'automatisation de la pizzeria, il vous est demandé de réaliser la même chose pour une pâtisserie spécialisée dans les tartes. On remplacera simplement le temps de repos de la pâte à pizza par le temps de pétrissage et de cuisson de la pâte (pâte sablée, feuilletée, brisée, à pain, à filo, à feuilles de brick ...).

Figure 5.1 – Source Sugar Mama

5.1.1 Livrable attendu

- 1 fichier SQL qui reprendra l'ensemble de la création de la base de données.
- 1 programme Python qui affichera le contenu de l'ensemble des tartes avec leur composition.
- 1 explication de la structure utilisée

Le mode d'insertion des données dans la base est à votre libre choix (SQL ou programme ou les deux).

5.2 Consignes

- L'anonymat entraîne l'anonymisation de la note (0).
- Le brouillon n'est pas une option, toute copie ressemblant à un brouillon ne sera pas corrigée (0).
- Les commentaires ne sont pas des fioritures.
- La clarté et la précision des schémas sont essentielles.

5.3 Précisions

- De manière courante, une table commence toujours par son identifiant unique.
- Respecter les casses et les présentations des attributs des différentes tables.
- Un commentaire dans une table peut être ajouté à l'aide de la commande : `COMMENT 'Nom de la garniture'`

Chapitre 6

Pizza Requêtes

6.1 Préambule

L'ensemble de ces exercices est basé sur des requêtes SQL. Bien que certaines soient farfelues, le but est de vous faire composer des requêtes applicables à n'importe quel environnement.

6.2 Un plat toujours chaud

Travailler avec une base de données entraîne un certain nombre de règles obligatoires qui passent notamment par la sauvegarde et la restauration. Nous allons donc apprendre à sauvegarder / restaurer une base de données.

— Pour sauvegarder une base, on peut utiliser la commande `mysqldump`.
— Pour restaurer une base, on exécute mysql en redirigeant le fichier de sauvegarde en entrée

6.2.1 Exemple

```
mysqldump --no-tablespaces -u pizza -ppizzamdp FirstPizza > savFirstPizza.sql

mysql -u pizza -ppizzamdp
mysql> drop database FirstPizza;
Query OK, 2 rows affected (0.09 sec)

mysql> show databases;
+--------------------+
| Database |
+--------------------+
| information_schema |
+--------------------+
1 row in set (0.01 sec)

mysql> create database FirstPizza;
mysql> quit
Bye

mysql -u pizza -ppizzamdp FirstPizza < savFirstPizza.sql

mysql -u pizza -ppizzamdp
mysql> show databases;
mysql> quit
```

6.3 Warnings

Un warning représente un avertissement de MySQL, pour le comprendre, il est possible de à MySQL de l'expliciter.

```
mysql> show warnings;
+---------+------+-----------------------------------------------------------------+
| Level | Code | Message |
+---------+------+-----------------------------------------------------------------+
| Warning | 1366 | Incorrect integer value: '' for column 'val_test' at row 2 |
+---------+------+-----------------------------------------------------------------+
```

Plus d'informations : https://dev.mysql.com/doc/refman/5.0/en/show-warnings.html

6.4 Ajout de données

Nous allons compléter les informations de notre table à l'aide d'un petit programme Python qui va aller lire les informations dans un fichier txt (`garniture.txt`.

Tomates	1	Champignons	0.8
Gruyere	1.2	Merguez	0.5
Creme Fraiche	0.6	Brocolis	0.7
Oignons	1.1	Fromage de chevre frais	0.9
Courgettes	1	Mozzarella	0.3
Olives	1.2	Fontina	1
Parmesan	1.4	Aubergines	1.1
Poivrons rouges	1.15	Poivrons jaunes	1.1
Poivrons verts	0.5	Capres	2
Filet de truite	0.2	Pomme de terre	0.5
Chicorees rouges	1.8	Bacon	0.3
Ananas	0.4	Gorgonzola	0.7
Cerneaux de noix	1	Jambon	0.7
Asperges	0.4	Epinards	0.15
Oeuf	2.2	Anchois	0.3
Pomme	0.4	Poire	0.5
Chocolat	3	Crevettes	0.4
Raisins	0.4	Banane	0.3
Pruneaux	0.3	Abricots Secs	0.3
Fruits Rouges	0.4	Noix de coco	0.5
Ricotta	0.6	Orange	4
Saint Jacques	3.5	Porc Braise	3
Blanc de Poulet	0.1	Ail	1
Thon	2.5	Moules	3
Palourdes	0.5	Roquette	0.25
Piments	0.5	Miel	0.10

TABLE 6.1 : Prix des Garnitures (€)

Attention

☢ Ne pas changer l'ordre des éléments, vos ids seraient alors mélangés, et vos pizzas déconstruites. Après il vous est possible aussi de complexifier votre programme en faisant une recherche puis une insertion.

6.4.1 Modification de la table Garnitures

Nous désirons ajouter les prix de chaque garniture, nous allons donc modifier la table Garniture en SQL.

```sql
ALTER TABLE `Garniture` ADD `prix` REAL NOT NULL DEFAULT '0';
```

6.4.2 Marche à suivre

On considèrera que les identifiants des garnitures sont séquentiels (de 1 en 1).

- Sauvegarder votre base de données `mysqldump -u pizza -ppizzamdp FirstPizza > savFirstPizza.sql`
- Écrire une requête vous permettant de modifier l'un des prix de garnitures `UPDATE Garniture set prix="0.50" where id_Garniture=51`
- Écrire un programme **affichant** les requêtes que vous allez envoyer à la base de données.
- Finalisez le programme.
- Controlez quelques prix ...recommencez au besoin.

Listing 6.1 – `corr_ajoute_prix_garnitures_3.py`

```python
#!/usr/bin/python3
# -*- coding: utf-8 -*-
import sys
import mysql.connector

conn = mysql.connector.connect(
                          user='pizza',
                          password='pizzamdp',
```

```
                          host='localhost',
                          database='FirstPizza'
)

# Traitement des requêtes aussitôt
conn.autocommit = True

# Il est nécessaire de créer un objet de type Curseur pour réaliser des requêtes
curs = conn.cursor()

# On ouvre le fichier "garniture.txt"
f = open('prix_garniture.txt', 'r')


id = 1

for line in f:
   requete = 'update Garniture set prix="' + line.rstrip() + '"' + " where id_Garniture=" + str(id)
   print (requete)

   try:
      curs.execute(requete)

   except mysql.connector.Error as err:
      print ("Erreur %d: %s" % (e.args[0],e.args[1]))
      sys.exit(1)


   id+=1

curs.close()
conn.close()

f.close ()
```

6.4.3 Prix des pâtes

Exercice

✓ Modifier la table `pate` pour lui adjoindre le prix des pâtes

Classique	1
Aux herbes	2
Complete	1,50
Pomme de terre	1,25
Au lait	1,30
Farine de Mais	1,55

TABLE 6.2 : Prix des Pâtes

Attention

☢ Ne pas changer l'ordre des éléments, vos ids seraient alors mélangés, et vos pizzas déconstruites

1. On modifie la table `Pate` pour autoriser l'ajout des prix.

```
ALTER TABLE `Pate` ADD `prix` REAL NOT NULL DEFAULT '0';
```

2. On crée un fichier de valeurs.

```
1
2
1.5
1.25
1.30
1.55
```

3. On adapte notre programme.

Listing 6.2 – `corr_ajoute_prix_pates_3.py`

```
#!/usr/bin/python3
# -*- coding: utf-8 -*-
import sys
import mysql.connector

conn = mysql.connector.connect(
                          user='pizza',
                          password='pizzamdp',
                          host='localhost',
                          database='FirstPizza'
)

# Traitement des requêtes aussitôt
conn.autocommit = True

# Il est nécessaire de créer un objet de type Curseur pour réaliser des requêtes
curs = conn.cursor()

# On ouvre le fichier "garniture.txt"
f = open('prix_pate.txt', 'r')


id = 1

for line in f:
    requete = 'update Pate set prix="' + line.rstrip() + '"' + " where id_Pate=" + str(id)
    print (requete)

    try:
        curs.execute(requete)

    except mysql.connector.Error as err:
        print ("Erreur %d: %s" % (e.args[0],e.args[1]))
        sys.exit(1)


    id+=1

curs.close()
conn.close()

f.close ()
```

Règle

Après de grosses manipulations, sauvegarder vos données.

6.5 Requêtes simples

6.5.1 Interrogation

1. Afficher l'ensemble des noms de pizzas
2. Afficher l'ensemble des pâtes
3. Afficher l'ensemble des garnitures
4. Afficher les 10 premières garnitures (`LIMIT`)
5. Afficher les 15 garnitures suivantes (`LIMIT`)
6. Afficher le nombre de garnitures (COUNT)
7. Afficher les garnitures avec leurs prix par ordre alphabétique
8. Afficher les garnitures avec leurs prix par ordre inverse alphabétique
9. Afficher l'ensemble des garnitures commençant par un `a`
10. Afficher l'ensemble des garnitures terminant par un `s`
11.  Afficher l'ensemble des garnitures composées de plusieurs mots

12. Afficher le nombre de garnitures composées de plusieurs mots
13.  Rechercher tous les ingrédients dont le prix est identique (`HAVING COUNT (*) > 1`)
    Dans un premier temps, on comptera le nombre de prix identiques (`GROUP BY`).

```
select nom_Pizza from Pizza;
select nom_pate from Pate;
select nom_garniture from Garniture;
select nom_garniture from Garniture LIMIT 10;
select nom_garniture from Garniture LIMIT 10,15;
select count(*) from Garniture;
select nom_garniture,prix from Garniture order by nom_garniture asc;
select nom_garniture,prix from Garniture order by nom_garniture desc;
select nom_garniture,prix from Garniture where nom_garniture LIKE 'a%';
select nom_garniture,prix from Garniture where nom_garniture LIKE '%s';
select nom_garniture,prix from Garniture where nom_garniture LIKE '% %';
select count(*) from Garniture where nom_garniture LIKE '% %';
select count(*), nom_garniture from Garniture group by prix;
select count(*), prix from Garniture group by prix having count(*) >1;
```

6.5.2 Calcul

1. Retrouver le numéro du plus grand `id` dans la table `Garnitures`
2. Retrouver le prix de l'ingrédient le plus cher (`MAX`)
3. Retrouver le ou les ingrédients les plus chers en utilisant le prix ci-dessus
4.  Retrouver le ou les ingrédients les plus chers (sans utiliser le prix numéraire trouvé ci-dessus)
   — Même chose pour le ou les ingrédients les moins chers
   — Ajouter la garniture "Basilic" au prix de 0,05 Euros, réexécuter votre commande.

```
select max(id_garniture) from Garniture;
select max(prix) as "Prix maximum" from Garniture;
select nom_garniture from Garniture where prix = 4;
select nom_garniture, prix from Garniture where prix = (select max(prix) from Garniture);
select nom_garniture, prix from Garniture where prix = (select min(prix) from Garniture);
INSERT INTO `Garniture` (`nom_garniture` ,`id_Garniture` ,`prix`) VALUES ('Echalotte', NULL , '0.10');
select nom_garniture, prix from Garniture where prix = (select min(prix) from Garniture);
```

6.5.3 Mise à jour

Attention

 Avant de réaliser ces mises à jour, prendre soin de **sauvegarder la base de données**.

1. Remplacer `Fromage de chevre frais` par `Fromage de chevre` en utilisant l'identifiant unique
2. Remplacer `Saint Jacques` par `Pétoncles` **en n'utilisant uniquement le champs nom**
3. Modifier le prix associé aux pétoncles à 2,50 €

```
select * from Garniture where nom_garniture like 'fromage%';
update Garniture set nom_garniture = "Fromage de chevre" where id_garniture = 8;

update Garniture set nom_garniture = "Petoncles" where nom_garniture like "%saint%jacques";

update Garniture set prix = 2.5 where nom_garniture = "Petoncles";
```

6.6 Requêtes temporelles

1. Afficher l'ensemble des noms de pizzas dont le temps de repos est égal à 1h30
2. Lister l'ensemble des pâtes dont le temps de repos est inférieur à 1h30
3. Lister l'ensemble des pâtes dont le temps de repos est supérieur à 1h30
4. Liste l'ensemble des pâtes dont le temps de repos est compris entre 15 et 120 minutes (`between`)

```
select * from Pate where tps_repos = '01:30:00';
select * from Pate where tps_repos < '01:30:00';
select * from Pate where tps_repos > '01:30:00';
select * from Pate where ( tps_repos >= '00:15:00' AND tps_repos <= '02:00:00');
select * from Pate where ( tps_repos BETWEEN '00:15:00' AND '02:00:00');
```

6.7 Jointures

6.7.1 Exemple commenté

Avant d'aborder les exercices sur les jointures, nous allons utiliser une autre base de données nommée `panier` afin d'analyser le fonctionnement d'une jointure. Cette base vous sera donnée par l'instructeur.

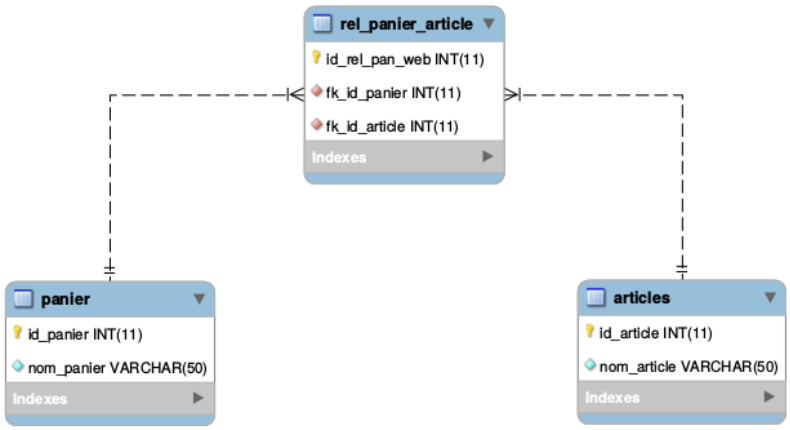


FIGURE 6.1 – Exemple pour la compréhension des jointures

Visualisons les données contenues dans chaque table.

```
mysql> select * from panier;
+-----------+-------------+
| id_panier | nom_panier |
+-----------+-------------+
| 1 | panier_eric |
| 2 | panier_tuxie |
+-----------+-------------+
2 rows in set (0.00 sec)

mysql> select * from articles;
+------------+---------------+
| id_article | nom_article |
+------------+---------------+
| 1 | papa pinguoin |
| 2 | maman pinguoin |
| 3 | mamie tuxie |
| 4 | papi tuxie |
+------------+---------------+
4 rows in set (0.01 sec)
```

```
mysql> select * from articles;
+------------+----------------+
| id_article | nom_article |
+------------+----------------+
| 1 | papa pinguoin |
| 2 | maman pinguoin |
| 3 | mamie tuxie |
| 4 | papi tuxie |
+------------+----------------+
4 rows in set (0.01 sec)
```

Nous souhaitons connaître le contenu de chaque panier. Pour se faire, il va falloir aller chercher tous les numéros d'articles dans la table `rel_panier_article` qui correspondent au panier désiré. Puis à l'aide de ces numéros aller trouver leur signification.

Une autre façon existe cependant, nous allons dans un premier temps associer l'ensemble des données entre elles.

```
mysql> select * from articles, panier, rel_panier_article;
+------------+----------------+-----------+--------------+---------------+--------------+--------------+
| id_article | nom_article | id_panier | nom_panier | id_rel_pan_web | fk_id_panier | fk_id_article |
+------------+----------------+-----------+--------------+---------------+--------------+--------------+
| 1 | papa pinguoin | 1 | panier_eric | 1 | 1 | 1 |
| 1 | papa pinguoin | 2 | panier_tuxie | 1 | 1 | 1 |
| 2 | maman pinguoin | 1 | panier_eric | 1 | 1 | 1 |
| 2 | maman pinguoin | 2 | panier_tuxie | 1 | 1 | 1 |
| 3 | mamie tuxie | 1 | panier_eric | 1 | 1 | 1 |
| 3 | mamie tuxie | 2 | panier_tuxie | 1 | 1 | 1 |
| 4 | papi tuxie | 1 | panier_eric | 1 | 1 | 1 |
| 4 | papi tuxie | 2 | panier_tuxie | 1 | 1 | 1 |
| 1 | papa pinguoin | 1 | panier_eric | 2 | 1 | 2 |
| 1 | papa pinguoin | 2 | panier_tuxie | 2 | 1 | 2 |
| 2 | maman pinguoin | 1 | panier_eric | 2 | 1 | 2 |
| 2 | maman pinguoin | 2 | panier_tuxie | 2 | 1 | 2 |
| 3 | mamie tuxie | 1 | panier_eric | 2 | 1 | 2 |
| 3 | mamie tuxie | 2 | panier_tuxie | 2 | 1 | 2 |
| 4 | papi tuxie | 1 | panier_eric | 2 | 1 | 2 |
| 4 | papi tuxie | 2 | panier_tuxie | 2 | 1 | 2 |
| 1 | papa pinguoin | 1 | panier_eric | 3 | 2 | 3 |
| 1 | papa pinguoin | 2 | panier_tuxie | 3 | 2 | 3 |
| 2 | maman pinguoin | 1 | panier_eric | 3 | 2 | 3 |
| 2 | maman pinguoin | 2 | panier_tuxie | 3 | 2 | 3 |
| 3 | mamie tuxie | 1 | panier_eric | 3 | 2 | 3 |
| 3 | mamie tuxie | 2 | panier_tuxie | 3 | 2 | 3 |
| 4 | papi tuxie | 1 | panier_eric | 3 | 2 | 3 |
| 4 | papi tuxie | 2 | panier_tuxie | 3 | 2 | 3 |
| 1 | papa pinguoin | 1 | panier_eric | 4 | 2 | 4 |
| 1 | papa pinguoin | 2 | panier_tuxie | 4 | 2 | 4 |
| 2 | maman pinguoin | 1 | panier_eric | 4 | 2 | 4 |
| 2 | maman pinguoin | 2 | panier_tuxie | 4 | 2 | 4 |
| 3 | mamie tuxie | 1 | panier_eric | 4 | 2 | 4 |
| 3 | mamie tuxie | 2 | panier_tuxie | 4 | 2 | 4 |
| 4 | papi tuxie | 1 | panier_eric | 4 | 2 | 4 |
| 4 | papi tuxie | 2 | panier_tuxie | 4 | 2 | 4 |
+------------+----------------+-----------+--------------+---------------+--------------+--------------+
32 rows in set (0.01 sec)
```

Puis extraire les éléments qui correspondent à ce que l'on souhaite c'est à dire les éléments. Pour se faire, observons l'entête des colonnes. **Un article Y est dans un panier X, si et seulement si,** `fk_id_panier` **est égal à X et** `fk_id_article` **est égal à Y.**
Ceci se traduit par :

```
mysql> select * from articles, panier, rel_panier_article where (( fk_id_panier = id_panier ) AND ( fk_id_article =
      id_article));
+------------+----------------+-----------+--------------+---------------+--------------+--------------+
| id_article | nom_article | id_panier | nom_panier | id_rel_pan_web | fk_id_panier | fk_id_article |
+------------+----------------+-----------+--------------+---------------+--------------+--------------+
| 1 | papa pinguoin | 1 | panier_eric | 1 | 1 | 1 |
| 2 | maman pinguoin | 1 | panier_eric | 2 | 1 | 2 |
| 3 | mamie tuxie | 2 | panier_tuxie | 3 | 2 | 3 |
| 4 | papi tuxie | 2 | panier_tuxie | 4 | 2 | 4 |
+------------+----------------+-----------+--------------+---------------+--------------+--------------+
4 rows in set (0.07 sec)
```

Il est bien sûr possible d'affiner la recherche en remplaçant l'`id` par une valeur spécifique et l'affichage en ne prenant en compte que ce qui nous intéresse.

```
mysql> select nom_article, nom_panier from articles, panier, rel_panier_article where ( (fk_id_panier = id_panier) AND
       ( fk_id_article = id_article) AND ( nom_panier = "panier_eric") );
+----------------+-------------+
| nom_article | nom_panier |
+----------------+-------------+
| papa pinguoin | panier_eric |
| maman pinguoin | panier_eric |
+----------------+-------------+
2 rows in set (0.00 sec)

mysql> select nom_article, nom_panier from articles, panier, rel_panier_article where (( fk_id_panier = id_panier )
      AND ( fk_id_article = id_article));
+----------------+--------------+
| nom_article | nom_panier |
+----------------+--------------+
| papa pinguoin | panier_eric |
| maman pinguoin | panier_eric |
| mamie tuxie | panier_tuxie |
| papi tuxie | panier_tuxie |
+----------------+--------------+
4 rows in set (0.00 sec)
```

6.8 Exercices sur les Jointures

1. Afficher les pizzas qui sont fabriquées avec de la pate `classique`
2. Afficher les pizzas avec leurs types de pâtes respectifs
3.  Afficher l'ensemble des ingrédients utilisés dans les pizzas (`IN`).
   On ne vous demande pas les noms de pizzas.
4. Afficher l'ensemble des garnitures qui ne sont pas utilisées dans vos pizzas (`NOT IN`).

```
select Pizza.nom_Pizza FROM Pizza WHERE ( Pizza.fk_id_Pate = ( SELECT Pate.id_Pate FROM Pate WHERE Pate.nom_pate = "
      Classique" ))
select Pizza.nom_Pizza, Pate.nom_Pate FROM Pizza, Pate WHERE ( Pate.id_pate = Pizza.fk_id_Pate );
select nom_garniture from Garniture where id_Garniture in ( select fk_id_Garniture from GarniturePizza );
select nom_garniture from Garniture where id_Garniture not in ( select fk_id_Garniture from GarniturePizza );
```

Chapitre 7

Pizzas améliorées

Notre base de données a été créée mais celle-ci nous apporte que peu de fonctionnalités à part celle du stockage. Nous allons maintenant découvrir ce qui différencie une base de données d'un environnement de stockage.

7.1 Clé primaire

Définition

✍ Dans une base de données relationnelle, une clé primaire est une contrainte d'unicité qui permet d'identifier de manière unique un enregistrement dans une table. Une clé primaire peut être composée d'un ou de plusieurs champs de la table. [1]

Exercice

✓ Vous avez précédemment créé 3 clés primaires composées d'un champs, identifiez les.

Les 3 clés primaires sont :

- `id_garniture` identifie de manière unique un élément de la table Garniture
- `id_pizza` identifie de manière unique un élément de la table Pizza
- `id_pate` identifie de manière unique un élément de la table Pate

Syntaxe

✎ Une clé primaire est spécifiée **lors de la création** par l'ajout de la commande `PRIMARY KEY (P_Id)`
Une clé primaire est spécifiée **après la création** par l'ajout de la commande `ALTER TABLE maTable ADD PRIMARY KEY (P_Id)`

Exercice

✓ À l'aide de la commande `DESCRIBE`, vérifiez la présence de ces clés primaires.

```
describe Garniture;
describe Pizza;
describe Pate;
```

7.2 Gestion des contraintes d'unicité

7.2.1 À l'aide du mot clé unique

Une garniture ne devrait pas apparaître deux fois dans la table. Nous allons donc le préciser.

```
ALTER TABLE `Garniture` ADD UNIQUE (`nom_garniture`);
```

Exercice

✓ Réalisez la même chose pour le nom des pizzas et des pates.

```
ALTER TABLE `Pizza` ADD UNIQUE (`nom_Pizza`);
ALTER TABLE `Pate` ADD UNIQUE (`nom_Pate`);
```

Nous allons maintenant tester la bonne prise en compte par MySQL de l'unicité des noms de pate.

Exercice

✓ Insérer dans la table `Pate` un nom existant.

1. Source : http://fr.wikipedia.org/wiki/Clé_primaire

```
INSERT INTO `Pate` SET nom_Pate="Classique";

ERROR 1062 (23000) at line 1: Duplicate entry 'Classique' for key 'nom_pate'
```

7.2.2 À l'aide d'une clé primaire

Une pizza est composée de différentes garnitures qui l'identifie de manière unique. L'ajout d'une pizza avec les mêmes ingrédients mais avec un nom différent ne devrait donc pas pouvoir être réalisé. Pour se faire, nous allons dire que le couple (référence Garniture, référence Pate) de la table `GarniturePizza` doit être unique. Ceci peut être réalisé en indiquant que ce couple forme une clé primaire.

Exercice

✓ Modifier la table pour prendre en compte cette nouvelle clé primaire.

```
ALTER TABLE `GarniturePizza` ADD PRIMARY KEY (fk_id_Pizza,fk_id_Garniture);
```

Nous allons maintenant tester la bonne prise en compte par MySQL de l'unicité de ce couple.

Exercice

✓ Insérer dans la table un couple de données existantes.

```
INSERT INTO `GarniturePizza` SET fk_id_Pizza = 5, fk_id_Garniture = 34;

ERROR 1062 (23000) at line 1: Duplicate entry '5-34' for key 'PRIMARY'
```

7.3 Clé étrangère

Définition

✍ Une clé étrangère, dans une base de données relationnelle, est une contrainte qui garantit l'intégrité référentielle entre deux tables. [1]

Syntaxe

✎ Une clé étrangère est spécifiée lors de la création par l'ajout de la commande `FOREIGN KEY (fk_Id) REFERENCES F_Table(F_Id)`
Une clé étrangère est spécifiée après la création par l'ajout de la commande `ALTER TABLE Ma_Table ADD FOREIGN KEY (fk_Id) REFERENCES F_Table(F_Id)`

Nous allons créer la clé étrangère entre la table `Pizza` et la table `Pate`.

```
   ALTER TABLE Pizza ADD FOREIGN KEY (fk_id_Pate) REFERENCES Pate (id_Pate);
```

Exercice

✓ Créer les contraintes de clés étrangères entre les tables Garniture / Pizza et la table PizzaGarniture.

```
ALTER TABLE GarniturePizza ADD FOREIGN KEY (fk_id_Pizza) REFERENCES Pizza (id_Pizza);
ALTER TABLE GarniturePizza ADD FOREIGN KEY (fk_id_Garniture) REFERENCES Garniture (id_Garniture);
```

Attention

☢ Sauvegarder la base de données.

Exercice

✓ Supprimer une garniture non utilisée dans vos pizzas. Que se passe t'il ?

```
DELETE FROM `FirstPizza`.`Garniture` WHERE `Garniture`.`nom_garniture` = "Fontina"
```

1. Source : http://fr.wikipedia.org/wiki/Clé_étrangère

La suppression de la garniture se passe sans problème.

Exercice

✓ Supprimer une garniture utilisée dans vos pizzas. Que se passe t'il ?

```
DELETE FROM `FirstPizza`.`Garniture` WHERE `Garniture`.`nom_garniture` = "Jambon"
```

La suppression de la garniture est interdite par la relation de clé étrangère.

Attention

☢ Restaurer la base de données.

7.4 Miam

Nous en avons maintenant terminé avec les fondementaux des bases de données. Ceci ne représente que la partie émergée de l'iceberg mais inutile de se lancer dans des choses plus compliquées si ces connaissances ne sont pas acquises.

Deuxième partie

Théorie des Bases de données

Chapitre 8

Cours

8.1 Introduction

Vous trouverez dans ces quelques pages un résumé de l'ensemble des notions vues précédemment dans ce cours.
Ces pages sont extraites et adaptées librement d'un document publié par Sébastien Namèche (http://sebastien.nameche.fr - sebastien@nameche.fr)

8.2 SQL & MySQL

Le langage SQL (Structured Query Language) est un langage de requête utilisé pour interroger des bases de données exploitant le modèle relationnel.

SQL fait l'objet d'une norme ANSI. Cependant, la quasi-totalité des serveurs de bases de données proposent des extensions qui rendent les programmes difficilement portables.

MySQL (dans sa version 3) implémente un sous-ensemble de la norme ANSI SQL92.

Les points forts de MySQL sont :

- implémentation libre et populaire ;
- facile à mettre en œuvre ;
- rapide à apprendre ;
- support multi-plateforme ;
- fiable et rapide.

8.3 Architecture

MySQL est basé sur une architecture client/serveur.

C'est-à-dire que les clients doivent s'adresser au serveur qui gère, contrôle et arbitre les accès aux données.

8.4 Objets

Un serveur MySQL gère une ou plusieurs base de données.
Chaque base de données contient différents types d'objets (tables, index, fonctions).
L'objet le plus représenté dans une base de données est la table.
Chaque table (appelées encore « relation ») est caractérisée par une ou plusieurs colonnes (ou « attributs »).

Le langage qui permet de gérer ces objets est appelé « Langage de Description des Données » (LDD)

Les données sont stockées dans les tables sous forme de lignes (ou « tuples »).

Le langage qui permet de manipuler les données est appelé « Langage de Manipulation des Données » (LMD).

8.5 Bases de Données

Les commandes `create database` et `drop database` permettent de créer ou de supprimer une base de données.

Leur syntaxe est simple. Attention lors de l'utilisation de drop database !

```
create database nombase
drop database nombase
```

L'utilisation de ces commandes n'est autorisée que si les droits de l'utilisateur connecté le permettent.

Dans l'utilitaire mysql, pour changer de base de données, utiliser la commande `use` comme ceci :

```
use nombase
```

Par exemple, un script de création d'une base de données commence souvent ainsi :

```
create database pizzas;
use pizza;
create table...
```

8.6 Notre Base de Données

Afin d'illustrer les éléments de ce cours, nous utiliserons le schéma de base de données suivant :

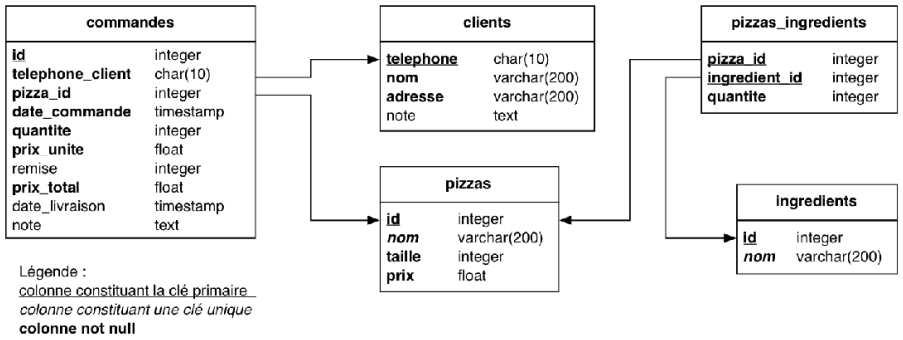


FIGURE 8.1 – Schéma Base de Données - Pizzas - Cours

8.7 Clés primaires et étrangères

Une table contient généralement une clé primaire.
Une clé primaire est constituée d'une ou plusieurs colonnes.

Les valeurs des colonnes qui constituent la clé primaire d'une table sont uniques pour toutes les lignes de la table. La clé primaire d'une table permet donc de faire référence de manière univoque à chaque ligne de la table.

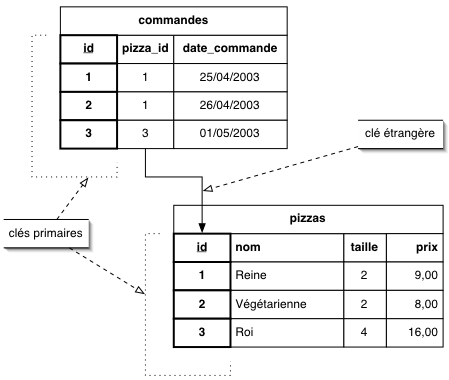


FIGURE 8.2 – Clés primaires et étrangères - Pizzas - Cours

Par exemple, les tables pizzas et commandes possèdent toutes deux une clé primaire qui est constituée de leur colonne id respective.
La colonne pizza_id de la table commandes fait référence à la colonne id (donc à la clé primaire) de la table pizzas.
Ce type de référence est appelée « **clé étrangère** ». Elle est implémentée uniquement sur certains moteurs de MySQL (InnoDB notamment).

8.8 LMD - Verbes

Le langage de manipulation des données (LMD) est constitué de quatre verbes :

```
SELECT INSERT UPDATE DELETE
```

Chacun de ces verbes possède une ou plusieurs clauses.
Lorsqu'elles sont entrées dans l'utilitaire `mysql`, les commandes SQL doivent se terminer par un point-virgule `;`. Dans d'autres interfaces le point-virgule final est généralement accepté mais pas souvent nécessaire.

Exemple :

```
select * from pizzas where nom='Roi';
```

La clause `from` permet de spécifier le nom de la table qui est interrogée, la clause `where` permet de préciser un critère de sélection.

Les chaînes de caractères sont entourées par des apostrophes simples. Si une apostrophe simple est présente dans la chaîne, elle doit être doublée. Par exemple : `'Il faut s''asseoir pour réfléchir !'`

8.8.1 LMD - SELECT

Le verbe `SELECT` permet de faire des requêtes sur une ou plusieurs tables. Il ne modifie jamais les données.

Une forme simple de `select` est :

```
select colonnes from tables where condition order by colonnes[asc/desc]
```

Par exemple, la requête suivante affiche le nom et le prix des pizzas pour deux personnes. Le résultat est trié par prix :

```
select nom, prix from pizzas where taille=2 order by prix;
```

Pour sélectionner toutes les colonnes d'une table, on utilise le caractère « * ».
Exemple :

```
select * from ingredients order by nom desc;
```

Il est également possible de réaliser des calculs. Exemple : combien coûtent trois pizzas reines ?

```
select prix*3 from pizzas where nom='Reine'
```

Les opérateurs de comparaison sont les suivants :
- arithmétiques : `= , < , > , <= , >= , !=`
- plage de valeur : `between valeur1 and valeur2`
- appartenance : `in (valeur1, valeur2, etc.)`
- nullité : `is null`, `is not null`
- comparaison de chaîne : `like 'pattern'`. Les caractères joker « % » et « _ » permettent respectivement de remplacer de 0 à n caractères (symbole %) et 1 caractère (symbole _).

Exemples

```
select * from commandes where quantite >= 2;
select nom, prix from pizzas where prix between 5 and 10;
select * from ingredients where id in (1, 2, 4);
select nom, note from clients where note is not null;
select telephone, nom from clients where telephone like '01%' order by nom;
```

Il existe aussi les 2 opérateur booléens classiques : `and` et `or`.

Exemple

```
select * from pizzas where taille = 4 and prix < 10;
```

Le verbe select dispose d'une clause qui permet de limiter le nombre d'enregistrements retournés : `limit`. Elle s'utilise ainsi :

```
select colonnes from table where condition limit [a,] b
```

Où a est l'index (à partir de 0) de la première ligne à afficher (0 si non précisé) et b est le nombre maximal de lignes. Si b est « -1 », toutes les lignes restantes sont retournées.

Par exemple, pour lister les lignes de la table commandes trois par trois, on utiliserait successivement :

```
select * from commandes limit 3;
select * from commandes limit 3, 3;
select * from commandes limit 6, 3;
...
```

Cette clause est particulièrement utile lorsque MySQL est utilisé pour afficher le contenu d'une table qui possède beaucoup de lignes (par exemple via une application Web)
Les fonctions de groupes sont : `count sum max min avg`.

Les fonctions de groupes permettent de répondre à des questions de type arithmétique.

Par exemple, la question « Quel est le nombre de clients de la pizzéria ? » s'écrirait ainsi :

```
select count(*) from clients;
```

Ces fonctions sont également appelées « fonctions agrégeantes » lorsqu'elles sont utilisées avec la clause group by.

Exemple

```
select telephone_client, sum(prix_total) from commandes group by telephone_client;
```

Il peut être pratique d'utiliser les alias de colonnes. Par exemple, pour établir le palmarès des trois pizzas les plus vendues :

```
select pizza_id as pizza_no, sum(quantite) as total from commandes group by pizza_id order by total desc limit 3
```

8.8.2 LMD - Jointure

Pour obtenir la liste **et** la quantité des pizzas commandées, il est nécessaire d'extraire des données provenant des tables commandes (quantités commandées) et pizzas (nom des pizzas). L'identifiant de la pizza (donnée présente dans les deux tables) formera le pivot de cette requête. Cette technique est appelée **jointure**.

```
select commandes.id, quantite, nom from commandes, pizzas where commandes.pizza_id = pizzas.id order by 1;
```

Le schéma ci-contre montre côte à côte un morceau du MPD (Modèle Physique des Données) de la base de données, le contenu des tables commandes et pizzas et le résultat de la requête ci-dessus.

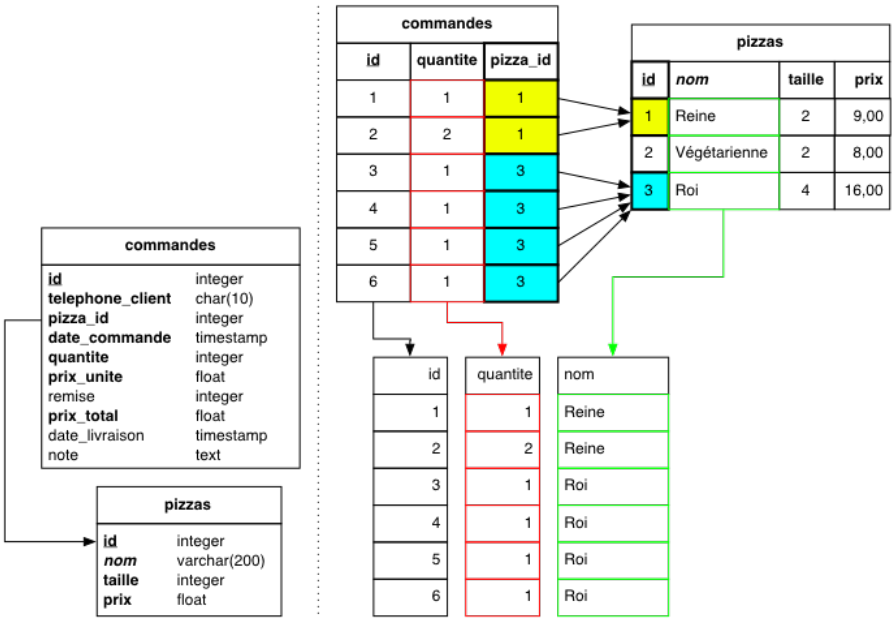


FIGURE 8.3 – Jointure - Pizzas - Cours

Il met en relief le fait que les jointures ne sont que la transcription des liens relationnels qui existent entre les tables de la base de données.

Il existe deux formes d'écriture pour les jointures :

```
select commandes.id, commandes.quantite, pizzas.nom from commandes, pizzas where commandes.pizza_id = pizzas.id
     order by 1;

select commandes.id, commandes.quantite, pizzas.nom from commandes  inner join pizzas on commandes.pizza_id =
     pizzas.id order by 1;
```

Ces deux requêtes fournissent le même résultat.

Si nous reprenons notre exemple de palmarès des pizzas vendues, il serait bien pratique d'afficher en première colonne le nom de la pizza plutôt que son identifiant. Ceci est possible en utilisant une jointure.

```
select nom, sum(quantite) as total from commandes, pizzas where commandes.pizza_id = pizzas.id group by pizza_id
     order by total desc limit 3;
```

Une jointure peut se faire avec plus de deux tables, ainsi, si nous souhaitons obtenir la liste et la quantité des ingrédients pour chaque pizza (afin de l'afficher en cuisine) :

```
select pizzas.nom as pizza, ingredients.nom as ingredient, pizzas_ingredients.quantite from pizzas, ingredients,
     pizzas_ingredients where pizzas_ingredients.pizza_id = pizzas.id and pizzas_ingredients.ingredient_id =
     ingredients.id order by 1;
```

Le nom de chaque colonne doit être préfixé par le nom de la table dont il est issu car il y aurait sinon ambiguïté. Il est également possible d'utiliser des alias pour les tables :

```
select piz.nom as pizza, ing.nom as ingredient, pzi.quantite from pizzas piz, ingredients ing, pizzas_ingredients
     pzi where pzi.pizza_id = piz.id and pzi.ingredient_id = ing.id order by 1
```

8.8.3 LMD - Insert

Le verbe `insert` permet d'ajouter des lignes à une table. Il s'utilise de deux manières :

```
insert into table (colonnes) values (valeurs)
insert into table (colonnes) select colonnes from tables
```

La liste des colonnes entre parenthèses est optionelle. Si elle est omise la liste des valeurs doit fournir une valeur pour toutes les colonnes de la table dans l'ordre dans lequel elles ont été spécifiées lors de la création de la table.
La seconde forme permet d'insérer dans la table de destination le résultat d'une requête.

Par exemple, ces deux commandes sont équivalentes **si** les colonnes `id` et `nom` sont ordonnées de cette manière dans la table ingredients.
L'ordre des colonnes est vérifiable par la commande `desc` :

```
mysql> desc ingredients;
+-------+--------------+------+-----+---------+----------------+
| Field | Type | Null | Key | Default | Extra |
+-------+--------------+------+-----+---------+----------------+
| id | int(11) | NO | PRI | NULL | auto_increment |
| nom | varchar(200) | NO | UNI | NULL | |
+-------+--------------+------+-----+---------+----------------+
```

```
insert into ingredients (id, nom) values (8, 'Poivron');
```

est équivalent à :

```
insert into ingredients values (8, 'Poivron');
```

Il n'est pas nécessaire de fournir une valeur pour les colonnes dites « nullable » (c'est-à-dire créer avec l'option not null).

Il est possible d'effectuer une insertion à partir du résultat d'une requête tout en fournissant certaines valeurs « externes ». Par exemple, pour créer une commande de deux pizzas « Roi » pour le client Alex Feinberg :

```
insert into commandes (telephone_client, pizza_id, date_commande, quantite, prix_unite, prix_total) select clients.
     telephone, pizzas.id, now(), 2, pizzas.prix, pizzas.prix*2 from pizzas, clients where pizzas.nom = 'Roi' and
     clients.nom = 'Alex Feinberg';
```

8.8.4 LMD - Update

Le verbe `update` permet de modifier des lignes déjà présentes dans une table.

Sa syntaxe est la suivante :

```
update table set colonne=valeur, colonne=valeur where condition
```

La clause `where` est optionelle. Si elle est omise, les modifications sont appliquées à la totalité des lignes de la table.

Par exemple, nous livrons la commande d'Alex Feinberg à 20h28 :

```
update commandes set date_livraison = '2003-05-14 20:28' where id = 7;
```

Il est possible d'introduire une formule dans la requête de mise à jour.

Par exemple, nous décidons d'augmenter de 2% le prix de toutes nos pizzas pour deux personnes :

```
update pizzas set prix = prix*1.02 where taille = 2;
```

8.8.5 LMD - Delete

Le verbe `delete` est utilisé pour supprimer des lignes d'une table.

Sa syntaxe est la suivante :

```
delete from table where condition
```

La clause `where` est optionelle. Toutes les lignes sont alors supprimées si elle est omise.

Dans une base de données relationnelle, il n'est pas possible de supprimer des lignes d'une table si des lignes d'une autre table font référence à l'une des valeurs de ces lignes (clé étrangère).

MySQL implémente plusieurs moteurs dont InnoDB et MyISAM.

— MyISAM ne prend pas en charge la gestion des contraintes des clés étrangères.
— InnoDB prend en charge cette gestion des contraintes des clés étrangères.

Dans le schéma ci-contre, seule la ligne surlignée en vert peut être supprimée sans risque pour la table pizzas.

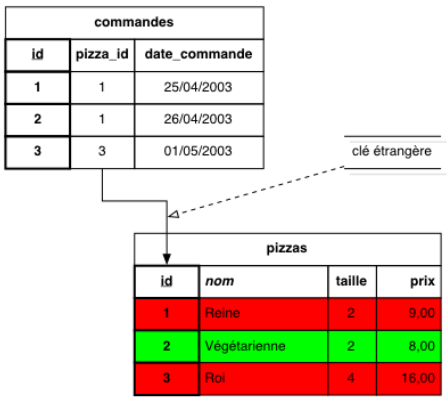


FIGURE 8.4 – Schéma Base de Données - Pizzas - Suppression

8.9 LDD, verbes

Le langage de description de données (LDD) est constitué de trois verbes : `CREATE DROP ALTER`
Ce langage permet de décrire les objets qui contiendront les données. Il ne sert pas à manipuler ces données. En général il n'est utilisé que lors de la création du schéma de la base de données au moment de l'installation du logiciel ou bien lors de la modification de ce schéma si des mises-à-jour sont effectuées.

8.9.1 LDD, Create Table

Le verbe `create table` est utilisé pour créer une table avec l'ensemble de ses colonnes.

Sa syntaxe est la suivante :

```
create table table
  ( colonne1 type [not null]
  , colonne2 type [not null]
  , ...
  , primary key (colonnes)
[ , unique (colonnes) ]
  , ... )
```

Les colonnes qui constituent la clé primaire doivent toutes être `not null`.

Les types de données les plus utilisés sont :

Type	Définition
varchar(n)	chaîne de caractères de taille variable
char(n)	chaîne de caractères de taille fixe
integer	entier
float	nombre à virgule flottante
date	date
time	heure
timestamp	date et heure (positionné à la date-heure actuelle si mis à NULL)
datetime	date et heure
text	textes longs
blob	valeur binaire longue
enum(liste)	énumération

TABLE 8.1 : Cours - Typage en MySQL

Il existe plusieurs modificateurs et types moins utilisés, consulter la documentation MySQL pour plus d'information.

8.9.2 LDD, Alter Table

La commande `alter table` permet de modifier certaines caractéristiques d'une table ou de l'une de ses colonnes. Elle est fréquemment utilisée pour ajouter une colonne à une table.
Par exemple, il nous est maintenant nécessaire d'ajouter la colonne livreur_id à la table commandes. Cette colonne contiendra l'identifiant du livreur ou la valeur NULL si le client est venu chercher la pizza sur place.

```
alter table commandes add column livreur_id integer;
```

La commande alter table permet également :

- d'ajouter une clé unique

  ```
  alter table table add unique (colonnes)
  ```

- de supprimer une colonne

  ```
  alter table table drop column colonne
  ```

- changer le nom et le type d'une colonne

  ```
  alter table table change column colonne nouveau\_nom nouveau\_type
  ```

- renommer une table

  ```
  alter table table rename as nouveau\_nom
  ```

8.9.3 LDD, Drop Table

La commande `drop table` supprime une table :

```
drop table table
```

8.10 Quelques fonctions usuelles

```
select user();
-> pizzeria@localhost
```

permet de connaître l'utilisateur connecté.

```
select substring_index( user(), '@', 1 );
-> pizzeria
```

découpe la chaîne en fonction du séparateur défini (ici @) puis retransmet le nombre d'éléments demandés du tableau ainsi défini (ici 1).

```
select version();
-> 5.41-0+wheezy1
```

permet d'obtenir la version actuelle de MySQL.

```
select sum(prix_total), round( sum(prix_total), 2 ) from commandes;
-> 167.59999847412 167.60
```

`sum` permet d'obtenir la somme d'une colonne.
`round` permet d'obtenir un arrondi d'un nombre flottant en précisant le nombre de chiffres après la virgule.

```
select concat( 'Pizza ', nom ) from pizzas;
-> Pizza Reine
   Pizza Roi
   Pizza Végétarienne
```

`concat` permet d'ajouter une chaîne de caractère.

```
update commandes set date_livraison = now() where id = 11;

select now();
+---------------------+
| now() |
+---------------------+
| 2015-02-22 15:14:34 |
+---------------------+
```

`now()` permet d'avoir la date/heure du jour.

```
select dayofweek(date_commande) as jour, count(*) as commandes from commandes group by dayofweek(date_commande) order
      by 2 desc;
-> jour commandes
   7 4
   6 2
   1 2
   2 1
   3 1
select dayofweek(now());
+------------------+
| dayofweek(now()) |
+------------------+
| 1 |
+------------------+
```

`dayofweek` permet d'avoir le jour de la semaine d'une date : 1=dimanche, 2=lundi, ..., 7=samedi

```
select nom, ifnull(note, '-pas de commentaire-') as note from clients;
-> nom note
   David Solomon -pas de commentaire-
   Linda Tchaïkowsky -pas de commentaire-
   Arthur Bab's Trés bon client
   Alex Feinberg Mauvais payeur
```

`ifnull` permet de renvoyer une chaîne de caractères dans la cas où le champs est `NULL`.

8.11 Index

Lorsque le moteur MySQL évalue une clause `where`, il doit parcourir l'ensemble des valeurs des colonnes concernées. Cette recherche est coûteuse en temps, aussi, afin d'améliorer la vitesse d'exécution des requêtes sur des tables importantes, on utilise des objets nommés `index`.
Un index est une relation simple entre l'ensemble des valeurs définies sur une colonne et les lignes de la table qui contiennent ces valeurs.

L'index est mis-à-jour en temps réel lorsque les lignes de la table sont modifiées par des requêtes `insert`, `update` ou `delete`. Par conséquent, **plus il y a d'index sur une table, moins l'enregistrement des modifications est rapide mais plus le gain en performance sur la clause `where` est efficace**.

Il est possible de créer un index sur une ou plusieurs colonnes ou de limiter son contenu à des valeurs uniques (en fait, les clés primaires et uniques sont implémentées sous forme d'index).

Les index sont spécifiés lors de la création d'une table ou en utilisant le verbe `alter table`.

Exemple

```
create table livres
( id integer unsigned not null primary key
, code_isbn varchar(20) not null unique
, titre varchar(200) not null
, auteur_id integer unsigned not null
, date_parution date not null
, resume text
, index (titre)
, index (auteur_id)
, index (date_parution) );
```

Cette requête de création de table va générer cinq index. Utiliser beaucoup d'index est cohérent dans ce cas car le nombre de modifications qui auront lieu sur cette table sera probablement bien inférieur au nombre de requêtes.

La commandes `show index` permet d'obtenir la liste de tous les index d'une table.

Exemple

```
mysql> show index from livres;
+--
      ------+------------+---------------+--------------+---------------+-----------+-------------+----------+--------+---------+

| Table | Non_unique | Key_name | Seq_in_index | Column_name | Collation | Cardinality | Sub_part | Packed | Comment |
+--
      ------+------------+---------------+--------------+---------------+-----------+-------------+----------+--------+---------+

| livres | 0 | PRIMARY | 1 | id | A | 0 | NULL | NULL | |
| livres | 0 | code_isbn | 1 | code_isbn | A | 0 | NULL | NULL | |
| livres | 1 | titre | 1 | titre | A | NULL | NULL | NULL | |
| livres | 1 | auteur_id | 1 | auteur_id | A | NULL | NULL | NULL | |
| livres | 1 | date_parution | 1 | date_parution | A | NULL | NULL | NULL | |
+--
      ------+------------+---------------+--------------+---------------+-----------+-------------+----------+--------+---------+

5 rows in set (0.00 sec)
```

Il est possible de nommer les index et les clés uniques lors de leur création :

```
create table livres
.../...
, unique code_isbn_uk (code_isdn)
, index titre_idx (titre)
.../...
```

Le verbe `alter table` permet de supprimer ou de créer des index sur une table a posteriori.

```
alter table table add index [nom_index] (colonnes)
alter table table add unique [nom_index] (colonnes)
alter table table drop index nom_index
alter table table drop primary key
```

Il existe également ces formes :

```
create [unique] index [nom_index] on table (colonnes)
drop index nom_index on table
```

8.12 Colonnes auto_increment

Très souvent les clés primaires des tables sont constituées d'une seule colonne dont le type est un entier. Il est alors très intéressant d'utiliser l'option `auto_increment` pour cette colonne lors de la création de la table.

Lors de l'insertion, si une colonne possède l'option `auto_increment` et si la valeur qui y est insérée est NULL, MySQL affecte automatiquement la prochaine valeur à cette colonne.

Exemple

```
create table messages (
   id integer unsigned not null auto_increment primary key,
   texte text not null
);

insert into messages (texte) values ('Hello World!');
insert into messages values (NULL, 'Salut et merci pour le poisson');
select * from messages;
-> id texte
    1 Hello World!
    2 Salut et merci pour le poisson
```

Il ne peut y avoir qu'une seule colonne `auto_increment` dans une table et cette colonne doit faire partie de la clé primaire.

8.12.1 Le dernier élément

La fonction `last_insert_id` permet de connaître la valeur qui a été donnée à cette colonne lors de l'insertion.

Exemple

```
insert into messages (texte) values ('Et pourtant elle tourne');
select last_insert_id();
+------------------+
| last_insert_id() |
+------------------+
| 3 |
+------------------+
1 row in set (0.00 sec)
```

8.13 Index Fulltext

Il est possible de créer des index « fulltext » sur les colonnes de type `char`, `varchar` ou `text` (et de ses dérivés : `tinytext`, `mediumtext` et `longtext`).

Ces index permettent de réaliser des recherches très rapides de mots sur des textes longs. L'algorithme utilise une technique dite de « scoring ».

Exemple

```
create table articles (
id integer unsigned not null auto_increment primary key
, titre varchar(200) not null
, contenu text not null
, fulltext (titre, contenu)
);
```

Les index suivants ont été créés :

```
mysql> show index from articles;
+--
      --------+------------+----------+--------------+-------------+-----------+-------------+----------+--------+----------+

| Table | Non_unique | Key_name | Seq_in_index | Column_name | Collation | Cardinality | Sub_part | Packed | Comment |
+--
      --------+------------+----------+--------------+-------------+-----------+-------------+----------+--------+----------+

| articles | 0 | PRIMARY | 1 | id | A | 0 | NULL | NULL | |
| articles | 1 | titre | 1 | titre | A | NULL | NULL | NULL | FULLTEXT |
| articles | 1 | titre | 2 | contenu | A | NULL | 1 | NULL | FULLTEXT |
+--
      --------+------------+----------+--------------+-------------+-----------+-------------+----------+--------+----------+

3 rows in set (0.00 sec)
```

Nous remplissons alors cette table avec des articles ce qui nous permet d'obtenir le résultat suivant :

```
mysql> select * from articles;
+----+-----------------------------+----------------------------------------+
| id | titre | contenu |
+----+-----------------------------+----------------------------------------+
| 0 | MySQL Tutorial | DBMS stands for DataBase ... |
| 1 | How To Use MySQL Efficiently | After you went through a ... |
| 2 | Optimising MySQL | In this tutorial we will show ... |
| 3 | 1001 MySQL Tricks | 1. Never run mysqld as root. 2. ... |
| 4 | MySQL vs. YourSQL | In the following database comparison ... |
| 5 | MySQL Security | When configured properly, MySQL ... |
+----+-----------------------------+----------------------------------------+
6 rows in set (0.00 sec)
```

Ces index nous permettent alors de faire une recherche basé sur un score de correspondance.

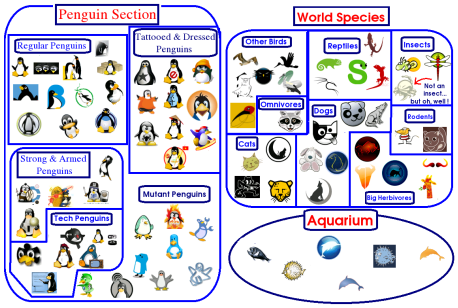
```
mysql> select *, match (titre, contenu) against ('database') as score from articles where match (titre, contenu)
      against ('database');
+----+-------------------+----------------------------------------+-----------------+
| id | titre | contenu | score |
+----+-------------------+----------------------------------------+-----------------+
| 4 | MySQL vs. YourSQL | In the following database comparison ... | 0.66266459031789 |
| 0 | MySQL Tutorial | DBMS stands for DataBase ... | 0.65545834044456 |
+----+-------------------+----------------------------------------+-----------------+
2 rows in set (0.00 sec)
```

Troisième partie

Étude de cas - Un Zoo

Chapitre 9

Zoo : Présentation du problème



Cet exercice est une adaptation libre d'un document de Stéphane Devismes (http://www-verimag.imag.fr/~devismes/BD/sujets.pdf).

Le directeur d'un zoo souhaite informatiser la gestion de son établissement.

Dans ce zoo, on trouve des animaux répertoriés par type (lion, léopard, girafe, escargot, ...).

Chaque animal possède un nom (Charly, Arthur, Enzo, ...) qui l'identifie de façon unique, une date de naissance et un pays d'origine.

On retient également les maladies que chaque animal a contracté depuis son arrivée au zoo.

Les animaux sont logés dans des cages. Chaque cage peut recevoir un ou plusieurs animaux. Certaines cages peuvent être inoccupées. Une cage correspond à une certaine fonctionnalité, qui n'est pas forcément liée à un type d'animal donné (par exemple une cage peut convenir à la fois aux girafes, aux éléphants et aux fauves, une autre aux grands oiseaux,...). Une cage est identifiée par un numéro, elle est située dans une allée, identifiée aussi par un numéro.

Des personnes sont employées par le zoo pour entretenir les cages et soigner les animaux. Chaque employé est identifié par son nom, et on connaît la ville où il réside. Il existe deux types de postes pour les employés : gardien ou responsable.

Chaque employé est affecté à un unique poste : soit gardien, soit responsable.
Un gardien s'occupe d'une ou plusieurs cages. Un responsable a en charge la surveillance de toutes les cages de une ou de plusieurs allées.

Une allée est sous la responsabilité d'un seul employé et toute cage occupée par au moins un animal est gardée par au moins un gardien ; les cages inoccupées ne sont pas gardées.

Exercice

✓ Modéliser l'ensemble des données à l'aide de tables

Un exemple de modélisation ...

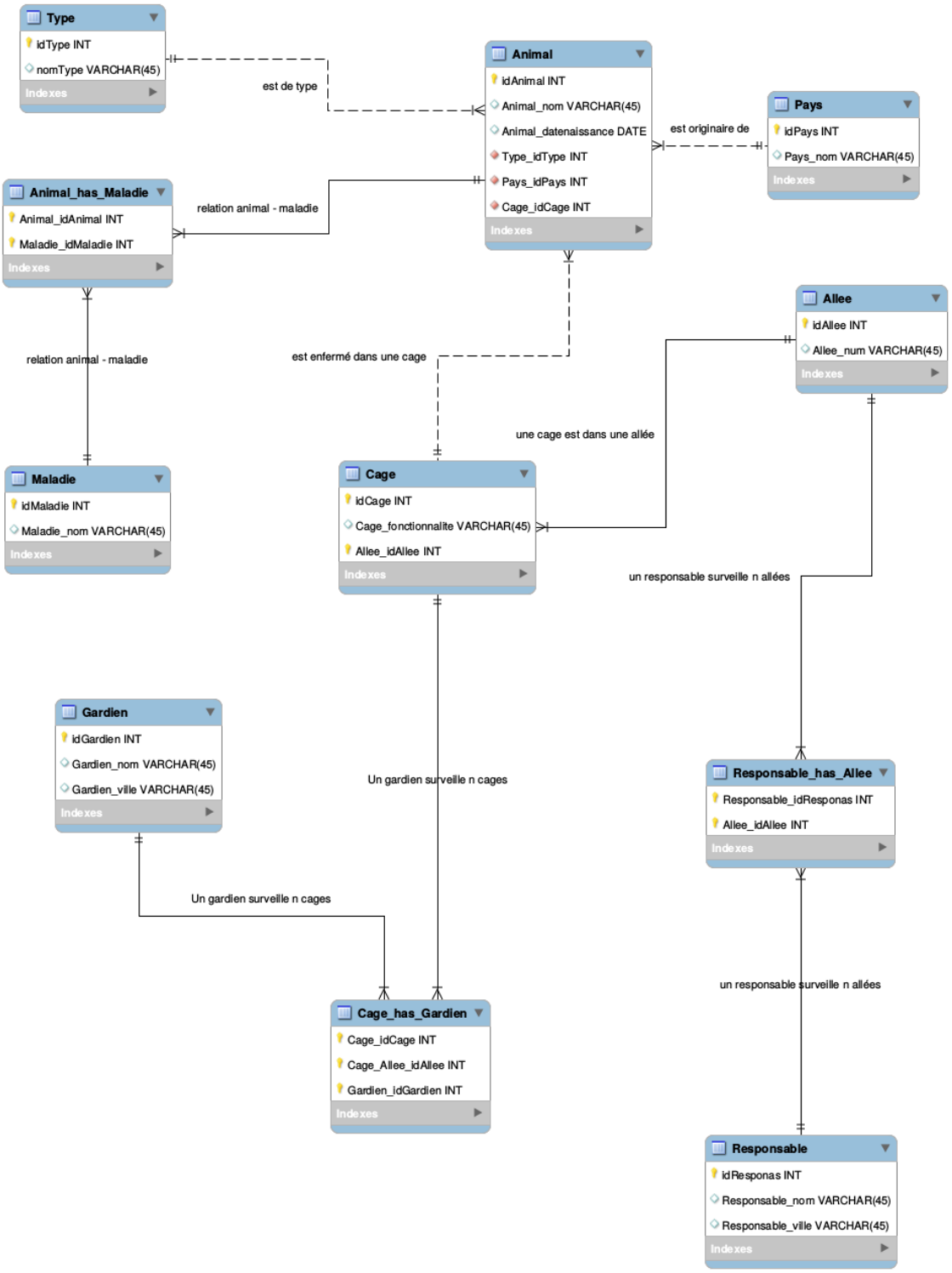


FIGURE 9.1 – Zoo - Exemple de Schéma Relationnel applicable au problème

Zoo : Schéma relationnel

10.1 Introduction

Le directeur du Zoo ayant une fibre informatique a réalisé lui même un schéma de données et vous demande de l'utiliser.

10.2 Tables

Les tables suivantes ont été créées et préalablement remplies afin de vous fournir un jeu d'essai.

nomA	sexe	type	pays	anNais	noCage
Charly	mâle	lion	Kenya	1990	12
Arthur	mâle	ours	France	1980	1
Chloé	femelle	pie	France	1991	3
Milou	mâle	léopard	France	1993	11
Tintin	mâle	léopard	France	1993	11
Charlotte	femelle	lion	Kenya	1992	12

Table 10.1 : Contenu de la table `Animaux`

noCage	Fonction	noAllée
11	enclos	10
1	fosse	1
2	aquarium	1
3	volière	2
4	grand aquarium	1
12	enclos	10

Table 10.2 : Contenu de la table `Cage`

noAllée	nomE
10	Peyrin
1	Adiba
2	Voiron

Table 10.3 : Contenu de la table `Responsables`

nomA	nomM
Charly	Rage de dents
Charly	Grippe
Milou	Angine
Chloé	Grippe

Table 10.4 : Contenu de la table `Maladies`

nomE	adresse
Peyrin	Nouméa
Berrut	Sartème
Sicard	Calvi
Voiron	Pointe à Pitre
Scholl	Ushuaïa
Adiba	Papeete

TABLE 10.5 : Contenu de la table `Employes`

noCage	nomE
11	Scholl
12	Berrut
11	Sicard
11	Berrut
1	Scholl
3	Scholl
12	Scholl

TABLE 10.6 : Contenu de la table `Gardiens`

Exercice

✓ Identifiez la clé primaire de chacune des tables (relations) de la base Zoo.

Les clés primaires apparaissent en soulignés.

Employes (nomE,adresse)
Responsables (noAllee,nomE)
Cages (noCage,fonction,noAllee)
Gardiens (noCage,nomE)
Animaux (nomA,sexe,type,pays,anNais,noCage)
Maladies (nomA,nomM)

TABLE 10.7 : Zoo - Clés primaires

Exercice

✓ Donnez le schéma relationnel de la base Zoo en spécifiant chacune des relations (1-1, 1-n, n-n), le type des données et les contraintes qui leurs sont associées.

Attribut	Domaine	Contraintes
nomE	chaîne de caractères	non null
adresse	chaîne de caractères	non nulle
noAllee	entier	entre 1 et 999, non nulle
noCage	entier	entre 1 et 999, non nulle
fonction	chaîne de caractères	non nulle
nomA	chaîne de caractères	
nomM	chaîne de caractères	
sexe	chaîne de caractères	femelle, mâle ou hermaphrodite. Valeur par défaut `mâle`
type	chaîne de caractères	non nul
pays	chaîne de caractères	Valeur par défaut ”France”
anNais	Entier	>= 1900

TABLE 10.8 : Zoo - Typage des attributs

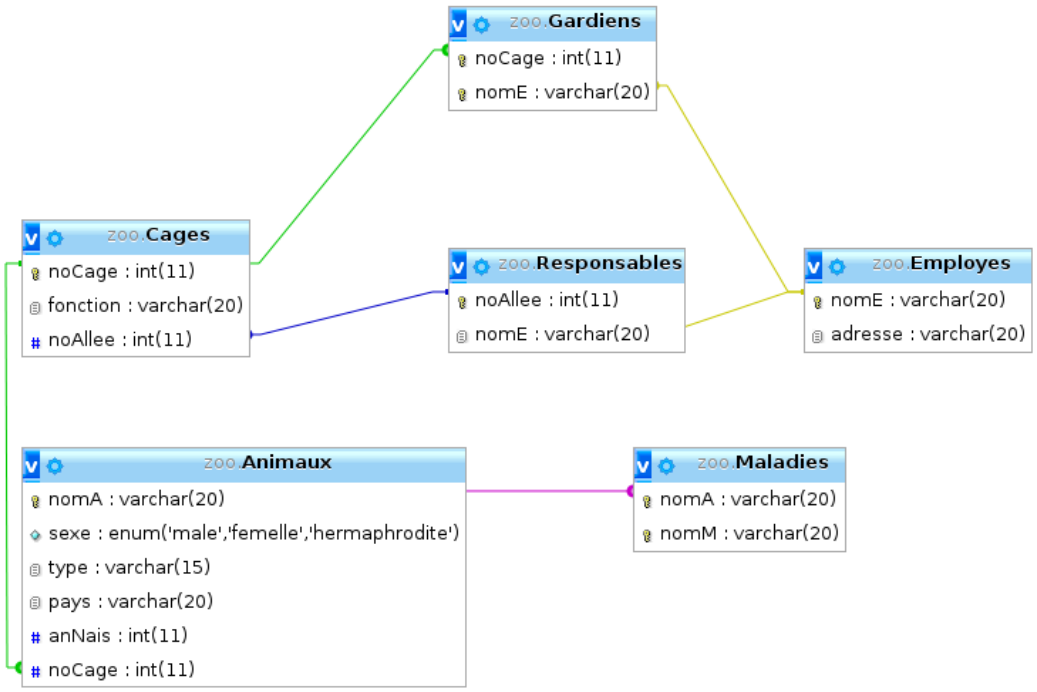


FIGURE 10.1 – Zoo - Schéma relationnel

Un clé étrangère ne peut être construite que si les tables qui sont pointées par ces clés existent. Nous allons donc créer notre base de données en 2 temps, dans un premier temps, les tables et les données, **dans un second temps les clés étrangères**.

Exercice

✓ Concevoir la base de données. On prendra en compte les clés étrangères dans la seconde partie de l'exercice

```
--
-- Base de donnees: `zoo`
--

DROP DATABASE IF EXISTS zoo;
CREATE DATABASE IF NOT EXISTS zoo;
USE zoo;

-- --------------------------------------------------------

DROP TABLE IF EXISTS Maladies ;
DROP TABLE IF EXISTS Animaux ;
DROP TABLE IF EXISTS Gardiens ;
DROP TABLE IF EXISTS Cages ;
DROP TABLE IF EXISTS Responsables ;
DROP TABLE IF EXISTS Employes ;

--
-- Structure de la table `Animaux`
--

CREATE TABLE IF NOT EXISTS `Animaux` (
   `nomA` varchar(20) NOT NULL DEFAULT '',
   `sexe` enum('male','femelle','hermaphrodite') NOT NULL DEFAULT 'male',
   `type` varchar(15) NOT NULL,
   `pays` varchar(20) DEFAULT 'France',
   `anNais` int(11) DEFAULT NULL,
```

```
   `noCage` int(11) NOT NULL,
   PRIMARY KEY (`nomA`)
) ENGINE=InnoDB;

--
-- Contenu de la table `Animaux`
--

INSERT INTO `Animaux` (`nomA`, `sexe`, `type`, `pays`, `anNais`, `noCage`) VALUES
('Arthur', 'male', 'ours', 'France', 1980, 1),
('Charlotte', 'femelle', 'lion', 'Kenya', 1992, 12),
('Charly', 'male', 'lion', 'Kenya', 1990, 12),
('Chloe', 'femelle', 'pie', 'France', 1991, 3),
('Milou', 'male', 'leopard', 'France', 1993, 11),
('Tintin', 'male', 'leopard', 'France', 1993, 11);

-- --------------------------------------------------------

--
-- Structure de la table `Cages`
--

CREATE TABLE IF NOT EXISTS `Cages` (
  `noCage` int(11) NOT NULL DEFAULT '0',
  `fonction` varchar(20) NOT NULL,
  `noAllee` int(11) NOT NULL,
  PRIMARY KEY (`noCage`)
) ENGINE=InnoDB;

--
-- Contenu de la table `Cages`
--

INSERT INTO `Cages` (`noCage`, `fonction`, `noAllee`) VALUES
(1, 'fosse', 1),
(2, 'aquarium', 1),
(3, 'voliere', 2),
(4, 'grand aquarium', 1),
(11, 'enclos', 10),
(12, 'enclos', 10);

-- --------------------------------------------------------

--
-- Structure de la table `Employes`
--

CREATE TABLE IF NOT EXISTS `Employes` (
  `nomE` varchar(20) NOT NULL DEFAULT '',
  `adresse` varchar(20) NOT NULL,
  PRIMARY KEY (`nomE`)
) ENGINE=InnoDB;

--
-- Contenu de la table `Employes`
--

INSERT INTO `Employes` (`nomE`, `adresse`) VALUES
('Adiba', 'Papeete'),
('Berrut', 'Sarteme'),
('Peyrin', 'Noumea'),
('Scholl', 'Ushuaia'),
('Sicard', 'Calvi'),
('Voiron', 'Pointe a Pitre');

-- --------------------------------------------------------

--
-- Structure de la table `Gardiens`
--

CREATE TABLE IF NOT EXISTS `Gardiens` (
  `noCage` int(11) NOT NULL DEFAULT '0',
```

```
  `nomE` varchar(20) NOT NULL DEFAULT '',
  PRIMARY KEY (`noCage`,`nomE`)
) ENGINE=InnoDB;

--
-- Contenu de la table `Gardiens`
--

INSERT INTO `Gardiens` (`noCage`, `nomE`) VALUES
(11, 'Berrut'),
(12, 'Berrut'),
(1, 'Scholl'),
(3, 'Scholl'),
(11, 'Scholl'),
(12, 'Scholl'),
(11, 'Sicard');

-- --------------------------------------------------------

--
-- Structure de la table `Maladies`
--

CREATE TABLE IF NOT EXISTS `Maladies` (
  `nomA` varchar(20) NOT NULL DEFAULT '',
  `nomM` varchar(20) NOT NULL DEFAULT '',
  PRIMARY KEY (`nomA`,`nomM`)
) ENGINE=InnoDB;

--
-- Contenu de la table `Maladies`
--

INSERT INTO `Maladies` (`nomA`, `nomM`) VALUES
('Charly', 'Grippe'),
('Charly', 'Rage de dents'),
('Chloe', 'Grippe'),
('Milou', 'Angine');

-- --------------------------------------------------------

--
-- Structure de la table `Responsables`
--

CREATE TABLE IF NOT EXISTS `Responsables` (
  `noAllee` int(11) NOT NULL DEFAULT '0',
  `nomE` varchar(20) NOT NULL,
  PRIMARY KEY (`noAllee`)
) ENGINE=InnoDB;

--
-- Contenu de la table `Responsables`
--

INSERT INTO `Responsables` (`noAllee`, `nomE`) VALUES
(1, 'Adiba'),
(10, 'Peyrin'),
(2, 'Voiron');
```

Exercice

✓ Intégrer les clés étrangères.

```
--
-- Base de donnees: `zoo`
--

USE zoo;

ALTER TABLE Responsables ADD FOREIGN KEY (nomE) REFERENCES Employes (nomE);
```

```
ALTER TABLE Cages ADD FOREIGN KEY (noAllee) REFERENCES Responsables (noAllee);

ALTER TABLE Gardiens ADD FOREIGN KEY (nomE) REFERENCES Employes (nomE);
ALTER TABLE Gardiens ADD FOREIGN KEY (noCage) REFERENCES Cages (noCage);

ALTER TABLE Animaux ADD FOREIGN KEY (noCage) REFERENCES Cages (noCage);

ALTER TABLE Maladies ADD FOREIGN KEY (nomA) REFERENCES Animaux (nomA);
```

10.3 Suppression des contraintes de clés étrangères

10.3.1 Identification des contraintes

Une clé étrangère est avant tout une contrainte ; il est donc nécessaire d'identifier le nom de la contrainte afin de pouvoir la supprimer. Plusieurs méthodes existe mais la plus simple est de demander comment a été créée la table qui supporte la contrainte.
Pour se faire, exécuter la commande suivante :

```
show create table nomTable;
```

Exercice

✓ Visualiser les contraintes associées à la table Animaux.

```
mysql> show create table Animaux;
+---------+---------------------+
| Table | Create Table |
+---------+---------------------+
| Animaux | CREATE TABLE `Animaux` (
  `nomA` varchar(20) NOT NULL DEFAULT '',
  `sexe` enum('male','femelle','hermaphrodite') NOT NULL DEFAULT 'male',
  `type` varchar(25) NOT NULL,
  `pays` varchar(20) DEFAULT 'France',
  `anNais` int(11) DEFAULT NULL,
  `noCage` int(11) NOT NULL,
  PRIMARY KEY (`nomA`),
  KEY `noCage` (`noCage`),
  CONSTRAINT `Animaux_ibfk_1` FOREIGN KEY (`noCage`) REFERENCES `Cages` (`noCage`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1 |
+---------+----------------------------+
1 row in set (0,00 sec)
```

10.3.2 Suppression d'une contrainte

Comme vous pouvez le constater, lors de la création d'une clé étrangère, un nom a été associé à la contrainte ... c'est ce nom qui sera utilisé pour supprimer la contrainte.
Pour se faire, exécuter la commande suivante :

```
alter table maTable drop foreign key nomContrainte;
```

Exercice

✓ Supprimer la contrainte associée à la table Animaux, constater, la remettre.

```
mysql> alter table Animaux drop foreign key Animaux_ibfk_1;
Query OK, 0 rows affected (0,04 sec)
Records: 0 Duplicates: 0 Warnings: 0
mysql> show create table Animaux;
+---------+---------------------+
| Table | Create Table |
+---------+---------------------+
| Animaux | CREATE TABLE `Animaux` (
  `nomA` varchar(20) NOT NULL DEFAULT '',
  `sexe` enum('male','femelle','hermaphrodite') NOT NULL DEFAULT 'male',
  `type` varchar(25) NOT NULL,
```

```
  `pays` varchar(20) DEFAULT 'France',
  `anNais` int(11) DEFAULT NULL,
  `noCage` int(11) NOT NULL,
  PRIMARY KEY (`nomA`),
  KEY `noCage` (`noCage`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1 |
+---------+---------------------------+
1 row in set (0,00 sec)
mysql> ALTER TABLE Animaux ADD FOREIGN KEY (noCage) REFERENCES Cages (noCage);
Query OK, 6 rows affected (0,11 sec)
Records: 6 Duplicates: 0 Warnings: 0
```

10.4 Compréhension des relations

Remarque

✓ Dans la suite de l'exercice, on considère que le champs `type` de la table `Animaux` est un `varchar (15)`.

Exercice

✓ Créer les requêtes nécessaires aux différentes assertions.

- Chloé, qui est au zoo depuis longtemps, vient de contracter une rage de dents en même temps que la grippe.
- Brigitte est une libellule femelle qui est arrivée au zoo il y a quelques jours de Tanzanie. Les libellules sont des animaux de type "insecte archiptere". Brigitte a été placée dans la cage numéro 3. Depuis son arrivée au zoo, elle n'a contracté aucune maladie. **Au besoin,** contrôler le type de "Brigitte" et corriger le.
- Chloe la Pie vient de manger Brigitte.
- Mme Bruandet, résidant à Papeete, vient d'être employée par le zoo. Elle a été affectée au gardiennage des cages 4, 11, et 12.
- La période d'essai n'a pas été profitable à Mme Bruandet qui quitte ses fonctions.
- On vient de construire une cage piscine, qui conviendra par exemple aux tortues. Elle a le numéro 40 et est située au bout de l'allée 10.

```
# Chloé, qui est au zoo depuis longtemps, vient de contracter une rage de dents en même temps que la grippe.
INSERT INTO `Maladies` (`nomA` , `nomM`) VALUES ('Chloe', 'Rage de dents');

# Brigitte est une libellule femelle qui est arrivée au zoo il y a quelques jours de Tanzanie. Les libellules sont des
      animaux de type "insecte archiptere". Brigitte a été placée dans la cage numéro 3. Depuis son arrivée au zoo,
     elle 'na contracté aucune maladie.
INSERT INTO Animaux VALUES ("Brigitte", "femelle", "insecte archiptere", "Tanzanie", 1981, 3);

# Contrôler le type de "Brigitte", corriger.
SELECT Type FROM Animaux WHERE ( nomA = "Brigitte");
# Le champs type a été fixé à 15 caractères il est insuffisant pour stocker l information.
ALTER TABLE `Animaux` CHANGE `type` `type` VARCHAR( 20 );
# Corriger le champs
UPDATE Animaux SET TYPE = "insecte archiptere" WHERE (nomA = "Brigitte");

# Chloe la Pie vient de manger Brigitte.
DELETE FROM Animaux WHERE (nomA = "Brigitte");

# Mme Bruandet, résidant à Papeete, vient 'dêtre employée par le zoo. Elle a été affectée au gardiennage des cages 4,
     11, et 12.
INSERT INTO Employes VALUES ("Bruandet", "Papeete");
INSERT INTO Gardiens VALUES (4, "Bruandet");
INSERT INTO Gardiens VALUES (11, "Bruandet");
INSERT INTO Gardiens VALUES (12, "Bruandet");

# La période d'essai n'a pas été profitable à Mme Bruandet qui quitte ses fonctions.
DELETE FROM Gardiens WHERE ( nomE = "Bruandet");
DELETE FROM Employes WHERE ( nomE = "Bruandet");

# On vient de construire une cage piscine, qui conviendra par exemple aux tortues. Elle a le numéro 40 et est située
     au bout de 'lallée 10.
INSERT INTO Cages VALUES ( 40, "piscine", 10);
```

10.5 Requêtes

Donnez la requête SQL, ainsi que le résultat de cette dernière pour les demandes suivantes :

1. Le nom des animaux du zoo.
2. Les différentes fonctions des cages présentes dans le zoo (**ne pas afficher les doublons**).
3. Les noms des léopards.
4. Les maladies contractées au moins une fois par des animaux du zoo.
5. Les noms et numéros de cage des animaux mâles qui sont originaires du Kenya et dont la date de naissance est antérieure à 1992.
6. Une requête produisant l'affichage suivant :

   — `Peyrin vit à Nouméa`
   — `Berrut vit à Sartène`
   — `Sicard vit à Calvi`
   — `Voiron vit à Pointe à Pitre`
   — `Scholl vit à Ushuaia`
   — `Adiba vit à Papeete`

   Aide

   ☞ Utilisez la commande `SELECT` et `CONCAT`

7. Le nom et l'âge des animaux en 2015.
8.  Le nom des **gardiens** qui habitent Ushuaia.
9.  La fonction des cages gardées par un / des employés habitants Calvi. Afin de contrôler votre requête, vous ajouterez un gardien "Eric" qui garde la cage numéro 3.
10. Les noms des animaux qui ont été malades
11. Les noms et types des animaux qui n'ont jamais été malades.
12. Les noms des animaux originaires du Kenya ayant déjà contractés une grippe.
13. Les numéros et fonctionnalités des cages qui sont inoccupées.
14. Donner pour chaque animal mâle l'ensemble des maladies qu'il a contracté.
15. Les noms des gardiens de Charly.
16. Le nom et le pays d'origine de l'animal doyen du zoo (il peut y en avoir plusieurs).

```
1) Le nom des animaux du zoo.
select nomA from Animaux;

2) Les differents types de fonctions associes aux cages dans le zoo.
select fonction from Cages group by fonction;

3) Les noms des leopards.
select nomA from Animaux where type = "leopard";

4) Les maladies contractees au moins une fois par des animaux du zoo.
Seules les maladies contractees sont enregistrees donc consultation de la table Maladie.
select nomM from Maladies group by nomM;

5) Les noms et numeros de cage des animaux males qui sont originaires du Kenya et dont la date de naissance est
     anterieure a 1992.
select nomA, noCage from Animaux where ( pays = "kenya" and anNais < 1992);

6) Une requete produisant l affichage suivant :
select concat (nomE, " vit a ", adresse) from Employes;

7) Le nom et l age des animaux en 2015.
select nomA, (2015 - anNais) from Animaux;
```

```
8) Le nom des gardiens qui habitent Ushuaia.
select Gardiens.nomE, Employes.adresse from Gardiens, Employes where ( Gardiens.nomE = Employes.nomE AND Employes.
      adresse = "ushuaia") group by Gardiens.nomE;

9) La fonction des cages gardees par un employe habitant Calvi.
select Cage.Fonction FROM Cage WHERE noCage in (select Gardiens.noCage from Gardiens WHERE Gardiens.nomE in (select
      Employes.nomE as nom_employe from Employes WHERE Employes.adresse = "Calvi"))

10) Les noms des animaux qui ont ete malades
select nomA from Maladies group by nomA;

11) Les noms et types des animaux qui n ont jamais ete malades.
select nomA from Animaux where nomA not in (select nomA from Maladies group by nomA);

12) Les noms des animaux originaires du Kenya ayant deja contractes une grippe.
select nomA from Animaux where (nomA in (select Maladies.nomA from Maladies where ( Maladies.nomM = "Grippe" )) and
      Animaux.pays="Kenya");

13) Les numeros et fonctionnalites des cages qui sont inoccupees.
select C.noCage, C.fonction from Cages as C where C.noCage not in (select Animaux.noCage from Animaux);

14) Donner pour chaque animal male l ensemble des maladies qu il a contractees (ensemble des couples nom d animal, nom
       de maladie).
select A.nomA, M.nomM from Animaux as A, Maladies as M where ( A.sexe="Male" AND A.nomA = M.nomA );

15) Les noms des gardiens de Charly.
select nomE from Gardiens where noCage = ( select noCage from Animaux where nomA = "Charly" )

16) Le nom et le pays d origine de l animal doyen du zoo (il peut y en avoir plusieurs).
select nomA, pays from Animaux where (anNais = ( select MIN(anNais) from Animaux ))
```

Chapitre 11

Zoo : Évolution des relations

Le but de cet exercice est de montrer les conséquences d'un mauvais choix dans la conception de la base de données.

11.1 Maladies

Un très grand nombre d'erreurs est apparu lors de la saisie des noms des maladies rendant inexploitables les statistiques qui leurs sont associées. Construire une table NomMaladies, Établir les relations nécessaires et écrire un programme qui permet la translation des informations de la table initiale vers la nouvelle table.

Attention

☢ Considérez que le nombre de maladies n'est pas de quelques unes mais de plusieurs centaines. Aussi, la migration ne peut être réalisée à la main.

La première étape est de créer une table qui hébergera le nom des maladies.

```
CREATE TABLE IF NOT EXISTS `NomMaladies` (
  `idNomMaladie` int(11) NOT NULL AUTO_INCREMENT,
  `nomMaladie` varchar(20) NOT NULL,
  PRIMARY KEY (`idNomMaladie`)
);
```

La seconde étape consiste à remplir cette table à partir de la table `Maladies`.

```
INSERT INTO NomMaladies (nomMaladie) SELECT `nomM` from Maladies GROUP BY nomM;
```

La troisième étape consiste à ajouter l'identifiant de maladie dans la table `Maladies` qui deviendra une clé étrangère.

```
ALTER TABLE `Maladies` ADD `fk_idNomMaladies` INT NOT NULL ;
```

La quatrième étape consiste à relier les anciennes données aux nouvelles données

Listing 11.1 – `maladies.py`

```
#!/usr/bin/python
# -*- coding: utf-8 -*-

import MySQLdb
import MySQLdb.cursors
import datetime

database = MySQLdb.connect(host = "localhost", user = "zoo", passwd = "zoo", db = "zoo", cursorclass=MySQLdb.cursors.
      DictCursor)
c = database.cursor()

c.execute("SELECT * FROM Maladies")
maladies = c.fetchall()

for maladie in maladies:
   print ("Nom : " + maladie ['nomM'])

   # On recherche la maladie dans la table des maladies
   c2 = database.cursor()
   c2.execute("SELECT idNomMaladie FROM NomMaladies WHERE nomMaladie = \"%s\"" % maladie ['nomM'])

   # Il ne peut y avoir qu'une maladie
   idMaladie = c2.fetchone()

   print ("Id de la maladie : %d " % idMaladie['idNomMaladie'])
   c2.close ()

   # On met à jour la table Maladies
   c3 = database.cursor()

   #~ print ("UPDATE Maladies SET fk_idNomMaladies = %d WHERE nomA=\"%s\" AND nomM=\"%s\"" % (idMaladie['idNomMaladie
         '], maladie['nomA'], maladie ['nomM']))
```

```
    c3.execute("UPDATE Maladies SET fk_idNomMaladies = %d WHERE ( nomA=\"%s\" AND nomM=\"%s\") " % (idMaladie['
        idNomMaladie'], maladie['nomA'], maladie ['nomM']))

    c3.close

database.commit()
```

Enfin, il ne reste plus qu'à supprimer les noms des maladies dans la table `Maladies` et lier la clé étrangère.

```
# On supprime la cle etrangere
ALTER TABLE Maladies DROP FOREIGN KEY Maladies_ibfk_1
# On supprime la cle primaire
ALTER TABLE Maladies DROP PRIMARY KEY
# On supprime la colonne nomM
ALTER TABLE `Maladies` DROP `nomM`;
# On reconstruit la cle etrangere
ALTER TABLE Maladies ADD FOREIGN KEY (nomA) REFERENCES Animaux (nomA);
# On reconstruit la cle primaire
ALTER TABLE `Maladies` ADD PRIMARY KEY (nomA, fk_idNomMaladies);
```

11.2 Horodatage

On souhaite maintenant pouvoir stocker pour chaque animal et pour chacune des maladies qu'il contracte, la date à laquelle il a contracté cette maladie. Quel est l'impact de cette modification sur le modèle relationnel précédent ? On ne vous demande pas de créer ce nouveau modèle.

La clé primaire composée par le nom de l'Animal et la Maladie ne peut plus convenir. On ajoute donc la date de la maladie dans la table Maladie et on joint ce champs à la clé primaire.

```
# On ajoute le champs dateM a la table Maladies
ALTER TABLE `Maladies` ADD `dateM` DATE NOT NULL;
# On supprime la cle etrangere
ALTER TABLE Maladies DROP FOREIGN KEY Maladies_ibfk_1;
# On supprime la cle primaire
ALTER TABLE Maladies DROP PRIMARY KEY;
# On reconstruit la cle etrangere
ALTER TABLE Maladies ADD FOREIGN KEY (nomA) REFERENCES Animaux (nomA);
# On reconstruit la cle primaire
ALTER TABLE `Maladies` ADD PRIMARY KEY(`nomA`,`fk_idNomMaladies`,`dateM`);
```

Quatrième partie

Sécurité

Les droits des utilisateurs

12.1 Préambule

Vous trouverez dans ces quelques pages un cours concernant la gestion des droits dans MySQL, ces pages sont extraites et adapté librement d'un document publié par Sébastien Namèche (http://sebastien.nameche.fr - sebastien@nameche.fr)

12.2 Droits d'accès

Le système de gestion des droits d'accès de MySQL est basé sur cinq tables de la base de données mysql (créées lors de l'installation de MySQL) :

```
mysql> show tables;
+---------------------------+
| Tables_in_mysql |
+---------------------------+
| columns_priv |
| db |
| event |
| func |
| general_log |
| help_category |
| help_keyword |
| help_relation |
| help_topic |
| host |
| ndb_binlog_index |
| plugin |
| proc |
| procs_priv |
| proxies_priv |
| servers |
| slow_log |
| tables_priv |
| time_zone |
| time_zone_leap_second |
| time_zone_name |
| time_zone_transition |
| time_zone_transition_type |
| user |
+---------------------------+
24 rows in set (0.00 sec)
```

Ces tables sont `user db host tables_priv columns_priv`. Dans le cadre de ce cours, nous nous limiterons à l'étude des deux premières.

Un utilisateur est identifié par le couple : `(user, host)`. La syntaxe utilisée est « user@host ».

Des privilèges sont associés à chaque utilisateur soit au niveau global, dans la table user, soit au niveau d'une base de données, dans la table db.

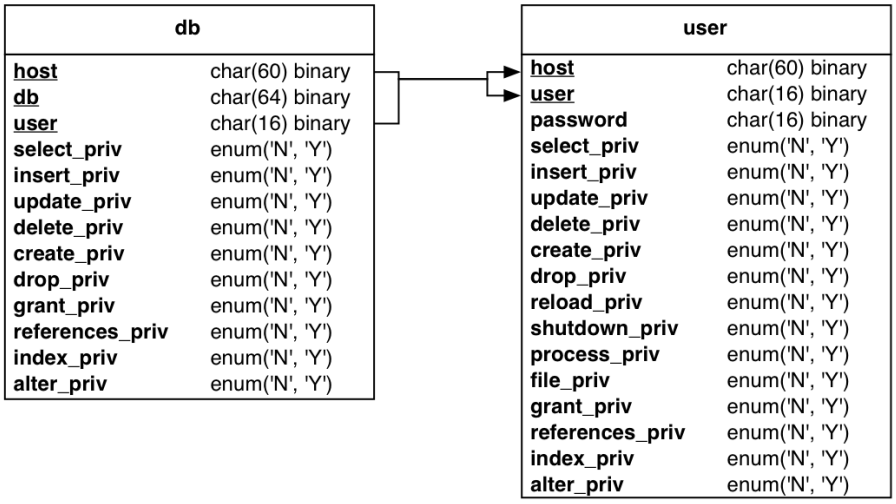


FIGURE 12.1 – Cours - Sécurité - Table User

La table user contient la liste des utilisateurs qui ont accès au serveur MySQL. Leur mot de passe est stocké dans la colonne password. Cette table contient également deux types de privilèges :

- les privilèges associés au serveur MySQL :
  `create(database) drop(database) reload shutdown process file`
- les privilèges associés aux tables de toutes les bases de données :
  `select insert update delete create(table,index) drop(table) - grant references index alter`

Ainsi, un utilisateur qui possède le privilège `create` dans la table `user` est autorisé à créer des bases de données et des tables dans toutes les bases de données. Il lui est également permis de créer des index dans toutes les bases de données mais uniquement lors de la création d'une table avec la commande `create table` (requis pour la création des clés primaires et uniques).

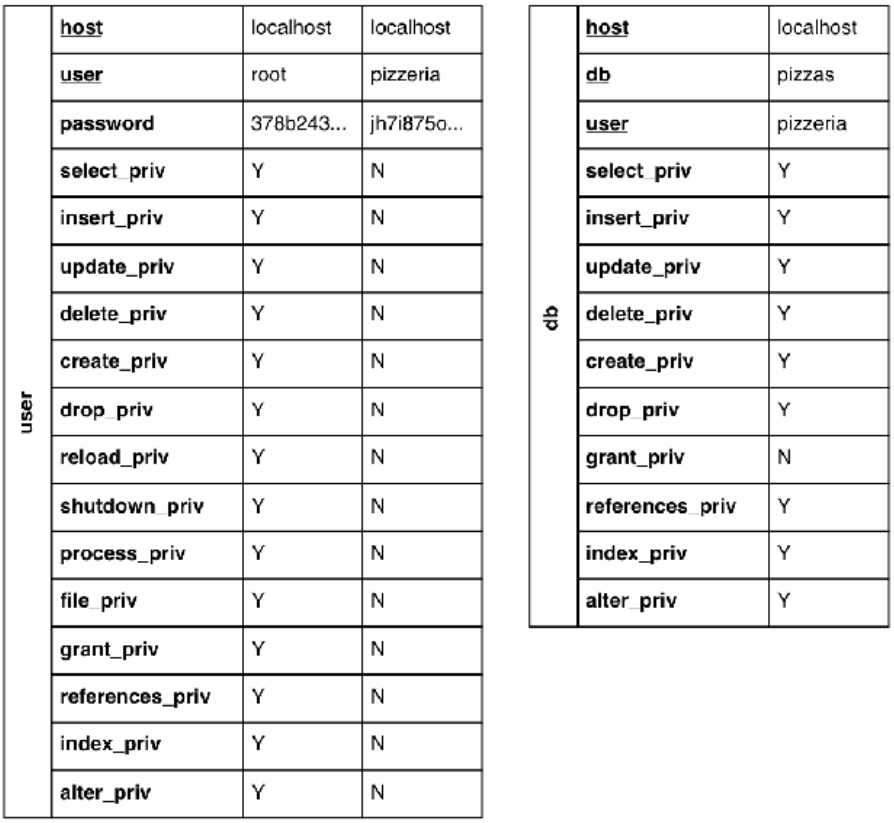
En général, il existe un seul utilisateur MySQL qui possède tous les droits au niveau global. Il est considéré comme l'administrateur du serveur MySQL. Bien souvent, il s'agit de « `root@localhost` ».

La table db précise les privilèges des utilisateurs pour chaque base de données. Les privilèges contenus dans la table sont pris en compte uniquement si le même privilège est « N » dans la table user pour le même couple `user@host`.

L'exemple ci-contre montre une configuration classique où l'utilisateur `root@localhost` possède tous les privilèges (il est considéré comme administrateur) et l'utilisateur `pizzeria@localhost` ne possède aucun privilège au niveau global et presque tous les privilèges sur une seule base de données.

Il existe deux méthodes pour gérer les utilisateurs et les privilèges qui y sont associés :

- modifier les tables `user`, `db`, etc. en utilisant le LMD ;
- utiliser les commandes SQL `grant`, `revoke` et `set password`.



user

host	localhost	localhost
user	root	pizzeria
password	378b243...	jh7i875o...
select_priv	Y	N
insert_priv	Y	N
update_priv	Y	N
delete_priv	Y	N
create_priv	Y	N
drop_priv	Y	N
reload_priv	Y	N
shutdown_priv	Y	N
process_priv	Y	N
file_priv	Y	N
grant_priv	Y	N
references_priv	Y	N
index_priv	Y	N
alter_priv	Y	N

db

host	localhost
db	pizzas
user	pizzeria
select_priv	Y
insert_priv	Y
update_priv	Y
delete_priv	Y
create_priv	Y
drop_priv	Y
grant_priv	N
references_priv	Y
index_priv	Y
alter_priv	Y

FIGURE 12.2 – Cours - Sécurité - Droits utilisateurs

La seconde méthode est privilégiée à la première.

Si l'une des tables `user, db, host, tables_priv ou column_priv` est modifiée, il est nécessaire d'utiliser l'instruction `flush privileges` afin que MySQL prenne en compte les changements.

12.2.1 Exemple

```
update user set password=password('Z') where user='root' and host='localhost';
flush privileges;
```

A contrario, la commande `flush privileges` n'est pas nécessaire lorsque l'on utilise les commandes `grant`, `revoke` et `set password`. Cette commande est équivalente aux deux commandes de l'exemple précédent :

```
set password for root@localhost = password('Z');
```

12.3 grant

La commande `grant` permet d'allouer des privilèges à un utilisateur. Celui-ci est créé s'il n'existe pas dans la table user. Sa syntaxe est :

```
   grant privileges on objets to user@host [identified by 'pass'] [with grant option]
```

Les clauses `identified by` et `with grant option` sont optionelles. La première permet de préciser un mot de passe pour l'utilisateur, la seconde l'autorise à donner ses privilèges à un autre utilisateur.

La liste `privileges` peut être remplacée par le mot « `all` » pour signifier tous les privilèges. Les privilèges sont séparés par des virgules et proviennent de cette liste : `select insert update delete create drop references index alter` à laquelle est ajoutée celle-ci pour les privilèges au niveau global (`on *.*`) : `reload shutdown process file`.

La liste objets peut prendre l'une de ces valeurs :

— « `*.*` » donne des privilèges au niveau global (table user) ;
— « `database.*` » affecte toutes les tables d'une base de données (table db) ;
— « `database.table` » affecte une table particulière (table tables_priv).

12.3.1 Exemple

Un script de création d'une base de données commence souvent ainsi :

```
create database pizzas;
grant select, insert, update, delete on pizzas.* to pizzeria@localhost identified by 'italia';
use pizzas;
create table...
```

Ce script est lancé par l'administrateur du serveur MySQL :

```
   mysql -u root -p mysql < cr_pizzas.sql
```

Ceci permet de créer un utilisateur spécifique avec des droits qui lui permettent uniquement d'utiliser le LMD afin de manipuler les données dans les tables que l'administrateur aura créées pour lui.

12.4 revoke

La commande `revoke` permet de révoquer les droits d'un utilisateur.
Sa syntaxe est :

```
revoke privileges on objets from user@host
```

La liste des privilèges et des objets sont utilisées de la même manière qu'avec la commande `grant`.

12.5 set password

La commande `set password` permet de modifier le mot de passe d'un utilisateur.
Sa syntaxe est :

```
set password for user@host = 'hash_of_password'
```

Elle est couramment utilisée avec la fonction `password` qui permet de crypter un mot de passe en clair vers le format utilisé par MySQL.

```
SET password FOR toto@localhost = password('titi');
```

12.5.1 Exemple

```
set password for pizzeria@localhost = password('PastaFantastica');
```

12.6 show grants

Pour en terminer avec la gestion des droits d'accès, l'une des options de la commande `show` permet d'obtenir une liste synthétique, mais exhaustive, des privilèges qui ont été octroyés à un utilisateur.
Sa syntaxe est :

```
show grants for user@host
```

12.6.1 Exemple

```
mysql> show grants for pizzeria@localhost;
+-----------------------------------------------------------------------------------------+
| Grants for pizzeria@localhost |
+-----------------------------------------------------------------------------------------+
| GRANT USAGE ON *.* TO 'pizzeria'@'localhost' IDENTIFIED BY PASSWORD '72de524a554dc726' |
| GRANT SELECT, INSERT, UPDATE, DELETE ON pizzas.* TO 'pizzeria'@'localhost' |
+-----------------------------------------------------------------------------------------+
2 rows in set (0.00 sec)
```

Chapitre 13

Bases de Données et Sécurité

13.1 Firewall et Serveurs

Cinquième partie

Pour aller plus loin

Triggers - Déclencheurs

14.1 Source

Ce chapitre s'inspire librement de l'article de John Cox : Introduction to MySQL Triggers (http://code.tutsplus.com/articles/introduction-to-mysql-triggers--net-12226).

14.2 Trigger / Déclencheur

Un trigger ou déclencheur permet d'effectuer une tâche lors de l'exécution d'un ordre de type `INSERT`, `UPDATE` ou `DELETE`.
Par exemple, si vous souhaitez tracer les modifications réalisées dans votre base de données, il est possible d'enregistrer toutes ces informations dans une table en créant un déclencheur qui va réaliser la tâche suivante : "À chaque fois qu'une ligne est modifiée, créer une nouvelle ligne dans une autre table pour m'indiquer qu'une mise à jour a été effectuée".

Un trigger peut être effectué avant (pour contrôler par exemple), ou après l'évènement défini. Ceci signifie que vous pouvez avoir un trigger avant une insertion et un autre après l'insertion d'une donnée dans votre table.

14.3 Utilisation simple

14.3.1 Exemple utilisé pour l'illustration

Afin d'illustrer l'utilisation des triggers, nous allons utiliser le concept d'un panier d'achats sur un site Web marchand. Un panier d'achats peut contenir des items. On y ajoute ou supprime des éléments.

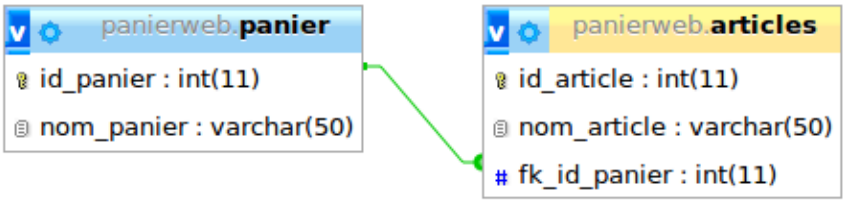


Figure 14.1 – Triggers - Base de données Exemple

Table 14.1 : Structure de la table panier

Colonne	Type	Null	Défaut	MIME
id_panier	int(11)	Non		
nom_panier	varchar(50)	Non		

Table 14.2 : Contenu de la table panier

id_panier	nom_panier
1	panier_eric

Table 14.2 : Contenu de la table panier (suite)

id_panier	nom_panier
1	panier_tuxie

Table 14.3 : Structure de la table articles

Colonne	Type	Null	Défaut	Relié à	MIME
id_article	int(11)	Non			
nom_article	varchar(50)	Non			
fk_id_panier	int(11)	Non	0	panier (id_panier)	

Table 14.4 : Contenu de la table articles

id_article	nom_article	fk_id_panier
1	dvd batman dark knight	1
2	bd swamp thing	1
3	t-shirt Superman	1
4	peluche tux	1
5	papa pinguoin	2
6	maman pinguoin	2

14.3.2 Abandon de panier

Imaginons que l'internaute ferme sa connexion alors qu'il avait mis des items dans son panier. Il faut alors vider le panier, et libérer les achats qu'ils souhaitaient réaliser. Pour se faire, il faut réaliser la séquence suivante : "Pour chaque achat du Panier, le supprimer puis Supprimer le panier".
En termes SQL, ceci peut se faire par 2 requêtes (dans le cas du panier 1) :

```
DELETE FROM `articles` WHERE `fk_id_panier` =1
DELETE FROM `panier` WHERE `id_panier` =1
```

Cette pratique pose le problème de la cohérence des données, si à un moment ou un autre on oublie l'une des deux requêtes (à cause d'un plantage système par exemple), la cohérence des données risque d'être interrompue. Pour éviter cela, nous allons créer un trigger de la façon suivante :

```
DELIMITER $$
CREATE TRIGGER `beforedeletepanier` BEFORE DELETE ON `panier`
 FOR EACH ROW DELETE FROM articles WHERE fk_id_panier = OLD.id_panier
$$
DELIMITER ;
```

Vous remarquerez la redéfinition du délimiteur SQL `DELIMITER` pour la chaîne de caractères `$$` le temps de la création du trigger. En effet, le délimiteur par défaut est `;` mais celui-ci sera utilisé pour permettre l'adjonction de plusieurs requêtes.
La commande de destruction d'un panier devient simplement :

```
DELETE FROM panier WHERE id_panier=1
```

14.3.3 Traçabilité

Nous souhaitons maintenant tracer toutes les insertions éffectuées au niveau du panier. Je désire donc stocker dans une table `traces` l'identifiant d'un article ainsi que l'identifiant du panier qui lui est associé.

Table 14.5 : Structure de la table traces

Colonne	Type	Null	Défaut	Relié à	MIME
id_traces	int(11)	Non			

Table 14.5 : Structure de la table traces (suite)

Colonne	Type	Null	Défaut	Relié à	MIME
fk_id_article	int(11)	Non		articles (id_article)	
fk_id_panier	int(11)	Non		panier (id_panier)	

Le fait que Tuxie ajoute à son panier (`panier_tuxie`), id numéro 2, l'article `mamie_pinguoin`, id numéro 7 se traduira en SQL de la façon suivante :

```
INSERT INTO `articles` (`id_article` ,`nom_article` ,`fk_id_panier`) VALUES ( '7', 'mamie_tuxie', '2')
INSERT INTO `traces` (`id_traces` , `fk_id_article` ,`fk_id_panier`) VALUES ( NULL, '7' , '2')
```

Il est possible de simplifier le traitement à l'aide d'un trigger :

```
DELIMITER $$
CREATE TRIGGER `afterinsertpanier` AFTER INSERT ON `articles`
 FOR EACH ROW INSERT INTO traces (`id_traces` , `fk_id_article` ,`fk_id_panier`) VALUES ( NULL, NEW.id_article , NEW.
      fk_id_panier)
$$
DELIMITER ;
```

```
INSERT INTO `articles` (`id_article` ,`nom_article` ,`fk_id_panier`) VALUES ( NULL, 'papi_tuxie', '2')
```

14.4 Utilisation évoluée

La structure de Bases de Données utilisée pour l'exemple ci-dessus n'est pas réellement fonctionnelle, la relation entre paniers et articles étant une relation de type n ↔ n, il est nécessaire d'utiliser une table de liaison.

Figure 14.2 – Triggers - Base de données Exemple 2

La construction de la base de données s'effectue donc comme suit :

```
CREATE TABLE `articles` (
  `id_article` int(11) NOT NULL AUTO_INCREMENT,
  `nom_article` varchar(50) NOT NULL,
  PRIMARY KEY (`id_article`)
);

CREATE TABLE `panier` (
  `id_panier` int(11) NOT NULL AUTO_INCREMENT,
  `nom_panier` varchar(50) NOT NULL,
  PRIMARY KEY (`id_panier`)
);

CREATE TABLE IF NOT EXISTS `rel_panier_article` (
  `id_rel_pan_web` int(11) NOT NULL AUTO_INCREMENT,
  `fk_id_panier` int(11) NOT NULL,
  `fk_id_article` int(11) NOT NULL,
  PRIMARY KEY (`id_rel_pan_web`),
  KEY `fk_id_article` (`fk_id_article`),
```

```
  KEY `fk_id_panier` (`fk_id_panier`)
);

--
-- Contraintes pour la table `rel_panier_article`
--
ALTER TABLE `rel_panier_article`
  ADD CONSTRAINT `rel_panier_article_ibfk_2` FOREIGN KEY (`fk_id_panier`) REFERENCES `panier` (`id_panier`),
  ADD CONSTRAINT `rel_panier_article_ibfk_1` FOREIGN KEY (`fk_id_article`) REFERENCES `articles` (`id_article`);

--
-- Contenu de la table `articles`
--

INSERT INTO `articles` (`id_article`, `nom_article`) VALUES
(1, 'papa pinguoin'),
(2, 'maman pinguoin'),
(3, 'mamie tuxie'),
(4, 'papi tuxie');


--
-- Contenu de la table `panier`
--

INSERT INTO `panier` (`id_panier`, `nom_panier`) VALUES
(1, 'panier_eric'),
(2, 'panier_tuxie');

--
-- Remplissage des paniers
--
INSERT INTO `rel_panier_article` (`id_rel_pan_web`, `fk_id_panier`, `fk_id_article`) VALUES
(1, 1, 1),
(2, 1, 2),
(3, 2, 3),
(4, 2, 4);
```

14.4.1 Exercices

Traduire les assertions suivantes en déclencheurs [1] :

— La destruction d'un panier entraîne la suppression des éléments associés dans la table `rel_panier_article`

— La destruction d'un article entraîne la suppression des éléments associés dans la table `rel_panier_article`

Du fait des contraintes associées au clés étrangères, il est nécessaire d'utiliser un déclencheur de type `BEFORE`.

```
DELIMITER $$
CREATE TRIGGER `beforedeletepanier` BEFORE DELETE ON `panier`
 FOR EACH ROW DELETE FROM rel_panier_article WHERE ( fk_id_panier = OLD.id_panier)
$$
DELIMITER ;
```

```
DELIMITER $$
CREATE TRIGGER `beforedeletearticle` BEFORE DELETE ON `articles`
 FOR EACH ROW DELETE FROM rel_panier_article WHERE ( fk_id_article = OLD.id_article)
$$
DELIMITER ;
```

14.5 Triggers en Cascade

De manière à illustrer la notion de `Cascade` des triggers nous allons rajouter une table à notre base de données.
Cette table se nommera `type_article` et contiendra les différents types d'articles proposés.
Les articles actuellement définis sont de type "Peluches", nous allons ajouter pour l'illustration un autre type, "Jeux de Société" et ajouter 2 jeux de société : "Echecs" et "Stratego".

1. Sauvegarder votre base de données avant !

Le panier d'Eric contiendra les 2 jeux, la panier de Tuxie uniquement "Stratego".

Exercice

✓ Réaliser les modifications demandées dans la base de données. Contrôler le contenu des deux paniers à l'aide d'une jointure.

```
# Creation de la table type_article
CREATE TABLE IF NOT EXISTS `type_article` (
  `id_type` int(11) NOT NULL AUTO_INCREMENT,
  `nom_type` varchar(50) NOT NULL,
  PRIMARY KEY (`id_type`),
  UNIQUE KEY `nom_type` (`nom_type`)
) ENGINE=InnoDB;

# Insertion de 2 elements dans la table type_article
INSERT INTO `panier`.`type_article` (`id_type`, `nom_type`) VALUES (NULL, 'Peluches'), (NULL, 'Jeux de Societe');

# Modification de la table articles
ALTER TABLE `articles` ADD `fk_type_article` INT NOT NULL DEFAULT '1';

# Contrainte par cle etrangère
ALTER TABLE `articles` ADD CONSTRAINT `rel_articles_type_ibfk_1` FOREIGN KEY (`fk_type_article`) REFERENCES `
     type_article` (`id_type`);

# Insertion de 2 elements dans la table articles
INSERT INTO `panier`.`articles` (`id_article`, `nom_article`, `fk_type_article`) VALUES (NULL, 'Stratego',2), (NULL, '
     Echecs',2);

# Le panier de Eric contient les 2 articles
INSERT INTO `panier`.`rel_panier_article` (`id_rel_pan_web`, `fk_id_panier`, `fk_id_article`) VALUES (NULL, '1', '5'),
      (NULL, '1', '6');

# Le panier de Tuxie contient le Stratego
INSERT INTO `panier`.`rel_panier_article` (`id_rel_pan_web`, `fk_id_panier`, `fk_id_article`) VALUES (NULL, '2', '5');

# Controle du contenu des paniers
SELECT nom_panier, nom_article FROM articles, panier, rel_panier_article WHERE (`fk_id_panier`= id_panier AND `
     fk_id_article`=id_article) ORDER BY nom_panier
```

Attention

☢ Sauvegarder votre base de données.

La société commerciale qui s'occupe de la vente décide de ne plus commercialiser les Peluches.
Pour se faire il faut donc, supprimer toutes les Peluches contenues dans les paniers.
Nous avons écrit précédemment un trigger prenant en compte la suppression d'un article.
La suppression d'un type d'article entraîne la suppression de tous les articles de ce type, qui lui même entraîne la destruction des éléments dans les paniers.

Exercice

✓ Créer le trigger associé, supprimer le type "Peluche", contrôler le résultat.

```
DELIMITER $$
CREATE TRIGGER `beforedeletetype` BEFORE DELETE ON `type_article`
 FOR EACH ROW DELETE FROM articles WHERE ( fk_type_article = OLD.id_type)
$$
DELIMITER ;
```

```
DELETE FROM `panier`.`type_article` WHERE `type_article`.`id_type` = 2
```

```
# Controle du contenu des paniers
SELECT nom_panier, nom_article FROM articles, panier, rel_panier_article WHERE (`fk_id_panier`= id_panier AND `
     fk_id_article`=id_article) ORDER BY nom_panier
```

Note

Il est possible d'utiliser le mot clé CASCADE avec les clés étrangères pour éviter les triggers mais ceci est une autre histoire...
http://www.mysqltutorial.org/mysql-on-delete-cascade/

14.6 Pizza

14.6.1 Suppression d'un type de Pâte

La suppression d'un type de Pâte doit entraîner la suppression de toutes les pizzas qui utilisent ce type de Pâte. Écrire le trigger associé à cette assertion.

```
DELIMITER $$
CREATE TRIGGER `before_delete_pate` BEFORE DELETE ON `Pate`
FOR EACH ROW
   DELETE FROM Pizza WHERE ( Pizza.fk_id_Pate = OLD.id_Pate);
$$
DELIMITER ;
```

14.7 Retour au Zoo

Quelles sont les incidences de la suppression d'une entrée dans la table `Employes` ?
Vous identifierez deux cas :

1. L'employé n'est pas responsable d'une allée
2. L'employé est responsable d'une allée

Écrire le trigger nécessaire pour permettre la suppression d'un employé. Ce dernier pourra renvoyer une exception dans le cas d'une incohérence.
Expliquer...

Note

Pour exécuter plusieurs actions au sein d'un trigger, il est nécessaire d'utiliser les mots clés `BEGIN` et `END`

La suppression d'un employé entraîne la suppression de cet employé en tant que Gardiens ET en tant que Responsable s'il l'est.

```
DELIMITER $$
CREATE TRIGGER `before_delete_employe` BEFORE DELETE ON `Employes`
FOR EACH ROW
   BEGIN
      DELETE FROM Gardiens WHERE ( Gardiens.nomE = OLD.nomE);
      DELETE FROM Responsables WHERE ( Responsables.nomE = OLD.nomE);
   END
$$
DELIMITER ;
```

Une exception apparaît lors de la suppression d'un employé qui est Responsable. En effet **1 responsable** est associé à **1 allée** qui est elle même associée à **une ou plusieurs cages**. Pour enlever un responsable, il faut lui enlever la surveillance d'une allée, il faut donc que cette allée soit dépourvue de cages ...

Une solution consisterait à permettre d'avoir plusieurs responsables pour une allée ou d'exercer moins de contraintes dans la base de données.

Procédures Stockées

15.1 Source

Ce chapitre s'inspire librement du tutoriel de Ankur Kumar Singh : Mysql Stored Procedure Tutorial (www.techflirt.com).

15.2 Définition d'une procédure stockée

Une procédure stockée est un ensemble de code SQL qui est enregistré dans la base de données et qui peuvent être invoquées par un programme, un trigger ou une procédure stockée elle-même. Une procédure stockée permet d'effectuer une liste de tâches directement sur la base de données, en d'autres termes, une procédure stockée permet de réaliser directement la logique de programmation dans la base.

15.3 Inconvénient d'une procédure stockée

Les principaux inconvénients d'une procédure stockée sont :
— la difficulté à pouvoir débugger
— le non support de la récursivité

15.4 Hello World!

Voici un premier exemple de procédure stockée, qui affiche un petit message. Attention à ne pas oublier les `()` dans la déclaration d'une procédure stockée.

```
DELIMITER $$

CREATE
PROCEDURE `helloworld`()
BEGIN
   select 'hello test';
END$$

DELIMITER ;
```

L'appel de cette procédure se fait de la manière suivante :

```
CALL helloworld();
```

La suppression de cette procedure se fera à l'aide de la commande `CALL`

```
DROP PROCEDURE IF EXISTS `helloworld`
```

La consultation de la liste des procédures stockées se fera à l'aide le commande `show procedure`

```
show procedure status;
```

15.5 Mise en application - Base de données Pizzas

Exercice

✓ Écrire une procédure stockée qui affiche la **somme des prix des ingrédients** pour la pizza 1 (ou autre si cette dernière n'existe plus)

La jointure peut s'écrire ainsi :

```
SELECT Garniture.prix, Pizza.id_Pizza from Pizza, Garniture, GarniturePizza WHERE ( Pizza.id_Pizza=1 AND
     GarniturePizza.fk_id_Pizza=1 AND GarniturePizza.fk_id_Garniture=Garniture.id_Garniture)
```

Ce qui permet d'obtenir la commande SQL suivante pour calculer le prix :

```
SELECT SUM (prix) FROM (SELECT Garniture.prix as prix from Pizza, Garniture, GarniturePizza WHERE ( Pizza.id_Pizza=1
     AND GarniturePizza.fk_id_Pizza=1 AND GarniturePizza.fk_id_Garniture=Garniture.id_Garniture)) total
```

La procédure stockée se définira donc de la façon suivante :

```
DELIMITER $$

CREATE
PROCEDURE `prixgarnitures`()
BEGIN
   SELECT SUM(prix) FROM (SELECT Garniture.prix as prix from Pizza, Garniture, GarniturePizza WHERE ( Pizza.id_Pizza
        =1 AND GarniturePizza.fk_id_Pizza=1 AND GarniturePizza.fk_id_Garniture=Garniture.id_Garniture)) total;
END$$

DELIMITER ;
```

```
CALL `prixgarnitures` ();
```

15.6 Exemple d'usage des paramétres dans une procèdure stockèe

La procédure stockée précédemment créée n'est pas totalement fonctionnelle, 2 éléments sont nécessaires pour corriger celà :

1. le fait de pouvoir choisir la pizza
2. le fait de contrôler que le numéro de pizza est bien correct

Nous n'aborderons pas le second point, sachez simplement que le `if` existe dans les procédures stockées. Voci un petit exemple d'utilisation des paramètres ...

```
DELIMITER $$

CREATE
PROCEDURE `bonjour` (IN texte VARCHAR (45))

BEGIN
   select "hello ",texte;
END$$

DELIMITER ;
```

```
CALL bonjour("Eric");
```

Exercice

✓ Modifier la procédure précédente pour accepter un numéro de pizza

```
DELIMITER $$

CREATE
PROCEDURE `prixgarnitures`(IN num_pizza INT (11))
BEGIN
   SELECT SUM(prix) FROM (SELECT Garniture.prix as prix from Pizza, Garniture, GarniturePizza WHERE ( Pizza.id_Pizza=
        num_pizza AND GarniturePizza.fk_id_Pizza=num_pizza AND GarniturePizza.fk_id_Garniture=Garniture.id_Garniture)
        ) total;
END$$

DELIMITER ;
```

```
call prixgarnitures (1)
```

15.7 Usage de variables locales

La dernière étape dans la quête du prix de la pizza est d'ajouter le prix de la pâte. Pour ceci, le plus simple (et le plus clair) reste d'utiliser une variable locale dans une procédure stockée.

Une variable locale se définit et s'utilise de la façon suivante.

```
DELIMITER $$

CREATE PROCEDURE `variables`()
BEGIN
   DECLARE i INT (3);
   DECLARE j INT (9) DEFAULT 6;

   SET i = 2;
   SELECT "i=",i;

   SELECT "j=",j;
   SET j = 30;

   SET j=4;
   SELECT j;

   SET @nompizza = (SELECT nom_Pizza from Pizza LIMIT 1);
   SELECT @nompizza;
END$$

DELIMITER ;
```

La déclaration des variables se fait en début de procédure.

15.7.1 Étendue de visibilitè d'une variable

Une variable dispose de sa propre portée(périmètre), qui définit sa durée de vie.

— Si vous déclarez une variable **à l'intérieur d'une procédure stockée**, elle sera hors de la portée quand la déclaration de FIN (`END`) de la procédure stockée sera atteinte.

— Si vous déclarez une variable **à l'intérieur du bloc** `BEGIN` - `END`, elle sera hors de la portée si le END est atteint.

Vous pouvez déclarer deux variables ou plus avec le même nom si elles ont des portées différentes. En effet, une variable est seulement effective dans sa propre portée. Cependant, ce genre de pratique a tendance à obscurcir la lisibilité du code.

Une variable qui commence par un arobase (@) est une variable de session. Elle est disponible et accessible jusqu'à la fin de session MySQL.

MySQL : @variable vs variable. Whats the difference :
http://stackoverflow.com/questions/1009954/mysql-variable-vs-variable-whats-the-difference

Exercice

✓ Écrire une procédure stockée qui affecte le prix de la pâte de la pizza, dont le numéro est passé en paramètre, à une variable `prix_pate`, puis l'affiche.

```
DELIMITER $$

CREATE
PROCEDURE `prixpate`(IN num_pizza INT (11))
BEGIN
   SET @prix_pate = (SELECT prix FROM Pate WHERE id_PATE IN (SELECT fk_id_Pate FROM Pizza WHERE ( Pizza.id_Pizza =
        num_pizza)));
   SELECT "prix_pate=",@prix_pate;
END$$

DELIMITER ;
```

Malgré tout, l'usage que nous avons fait des procédures stockées ne nous permet pas de renvoyer une valeur. Voici donc un exemmple de syntaxe permettant de renvoyer une valeur.

```
PROCEDURE `prixpizza`(IN num_pizza INT (11), OUT prix_pizza DOUBLE)

CALL prixpizza(1,@prix);

SELECT @prix;
```

15.7.2 Prix total

Le prix de la pizza sera défini par : $prixpizza = prixdesgarnitures + prixdelapate$

Exercice

✓ Écrire une procédure qui prend en paramètre un numéro de pizza et retourne le prix de la pizza. On commencera par écrire une procédure qui donne le prix de la pâte associée à une pizza.

```
DELIMITER $$

CREATE
PROCEDURE `prixpate`(IN num_pizza INT (11))
BEGIN
   SET @prix_pate = (SELECT prix FROM Pate WHERE id_PATE IN (SELECT fk_id_Pate FROM Pizza WHERE ( Pizza.id_Pizza =
        num_pizza)));
   SELECT "prix_pate=",@prix_pate;
END$$

DELIMITER ;
```

```
DELIMITER $$

CREATE
PROCEDURE `prixpizza`(IN num_pizza INT (11), OUT prix_pizza DOUBLE)
BEGIN
   SET @prix_pate = (SELECT prix FROM Pate WHERE id_PATE IN (SELECT fk_id_Pate FROM Pizza WHERE ( Pizza.id_Pizza =
        num_pizza)));
   SELECT "prix_pate=",@prix_pate;

   SET @prix_garniture = (SELECT SUM(prix) FROM (SELECT Garniture.prix as prix from Pizza, Garniture, GarniturePizza
        WHERE ( Pizza.id_Pizza=num_pizza AND GarniturePizza.fk_id_Pizza=num_pizza AND GarniturePizza.fk_id_Garniture=
        Garniture.id_Garniture)) total);
   SELECT "prix_garniture=",@prix_garniture;

   SET prix_pizza = @prix_pate + @prix_garniture;
END$$

DELIMITER ;
```

```
mysql> CALL prixpizza(1,@prix);
+------------+------------+
| prix_pate= | @prix_pate |
+------------+------------+
| prix_pate= | 1 |
+------------+------------+
1 row in set (0.00 sec)

+-----------------+-----------------+
| prix_garniture= | @prix_garniture |
+-----------------+-----------------+
| prix_garniture= | 2.12 |
+-----------------+-----------------+
1 row in set (0.00 sec)

Query OK, 0 rows affected (0.00 sec)

mysql> SELECT @prix;
+-------+
| @prix |
+-------+
| 3.12 |
+-------+
1 row in set (0.00 sec)
```

PHP / MySQL

L'interface homme machine (IHM) que nous avons jusqu'alors utilisée via le langage de programmation Python est suffisant dans le cas d'un usage isolé ou de tâche mais largement insuffisant dans le cas d'un usage à multiples niveaux de responsabilités ou dans le cas d'un accès partagé. Dans ces cas de figure, l'IHM préférée est celle du Web.
Bien que Python puisse lui même créer des pags web, nous avons préféré introduire un nouveau langage à ce support de cours car c'est celui le plus utilisé sur la toile : le PHP.

16.1 Au commencement : le HTML

Le HTML est un langage de description formé de balise qui sont interprétés par le navigateur pour obtenir un résultat [1].

Listing 16.1 – `hello.html`

```
<HTML>
<HEAD>
<TITLE>Hello World</TITLE>
</HEAD>
<BODY>
<h1>Hello World</h1>
This is your first web page !<br>
That's all folk !<br>
</BODY>
</HTML>
```

Le défaut de ces pages HTML est qu'elles sont statiques et que si vous avez 150 descriptions à réaliser, cela signifie de créer 150 fichiers HTML. Pour permettre cette dynamique, deux solutions sont possibles : la première et la plus ancienne est d'utiliser les CGI (Common Gateway Interface), la seconde d'utiliser un langage approprié au serveur Web. Mais avant toute chose il faut comprendre le fonctionnement d'une page WEB.
Les deux illustrations sont extraites du site de l'Université de Paris 13 : http://moodle.iutv.univ-paris13.fr

1. Qui pourra être différent selon les navigateurs !

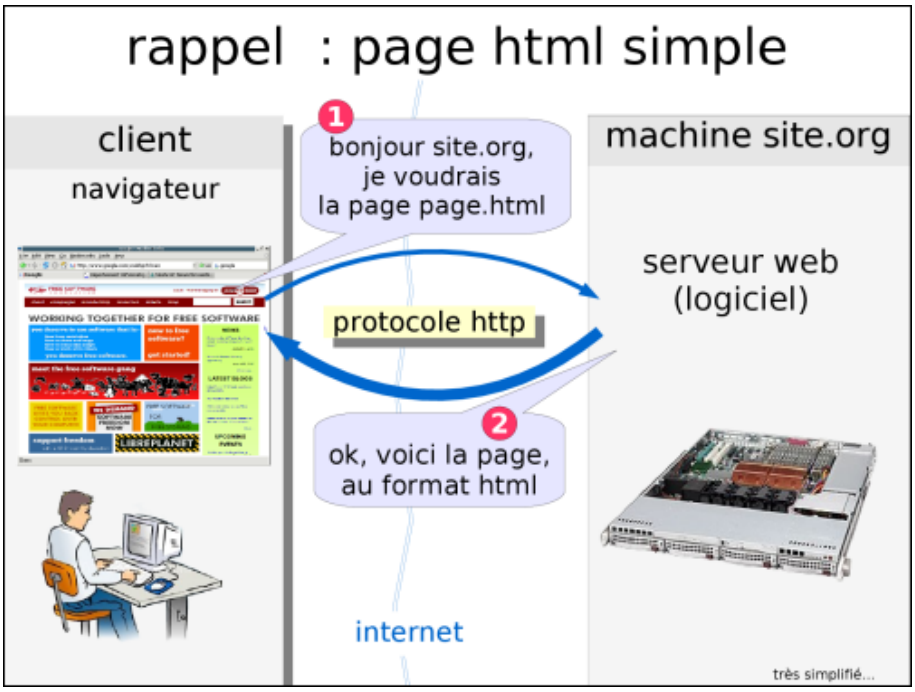


FIGURE 16.1 – Communication Client Serveur WEB (1/2)

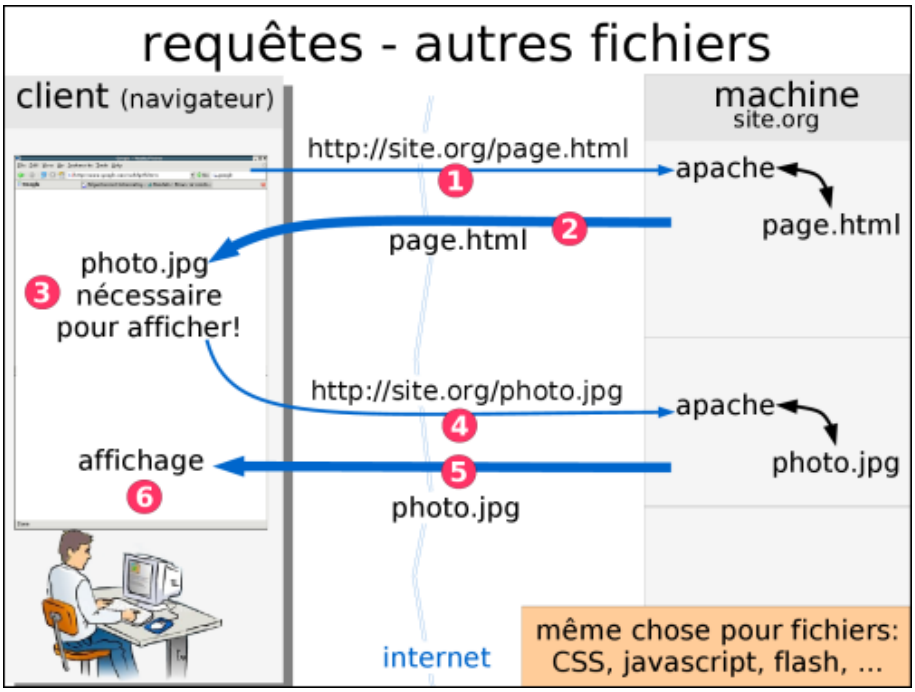


FIGURE 16.2 – Communication Client Serveur WEB (2/2)

Dans le cas de pages dynamiques, le serveur au lieu de redonner une page statique, va retransmettre le résultat d'un programme.

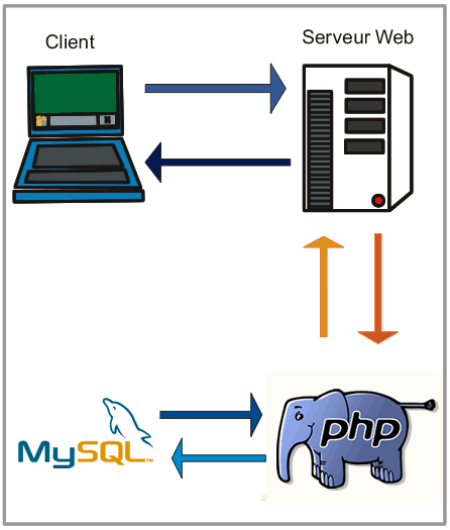


FIGURE 16.3 – Communication Client Serveur WEB LAMP/WAMP

16.2 CGI

Le répertoire associé aux fichiers CGI permet d'exécuter n'importe quel programme au travers du système et d'en retransmettre son résultat au navigateur. Dans le cas d'une Ubuntu, le répertoire `cgi-bin` se situe dans `/usr/lib`.
La seule contrainte est de débuter la page résultante par deux lignes vides.
Concernant le programme en lui-même, étant donné qu'il doit être exécuté, celui-ci doit avoir des droits d'exécutions (`chmod +x` sous Linux) et pouvoir être accéder par le compte qui exécute le programme serveur Web (`www-data` sous debian)

16.2.1 Exemples

Listing 16.2 – `liste.cgi`

```
#!/bin/sh

echo ""
echo ""

echo "<html>"
echo "<head>"
echo "<title>CGI Python</title>"
echo "</head>"
echo "<body>"

ls /etc

echo "</body>"
echo "</html>"
```

permettra d'exécuter la commande `ls` (liste des fichiers) sur le répertoire `/home` et d'en afficher le résultat sur une page Web.
`http://localhost/cgi-bin/liste.cgi`

Listing 16.3 – `listepy.cgi`

```
#!/usr/bin/python

print ("\n")
print ("<html>")
print ("<head>")
print ("<title>CGI Python</title>")
print ("</head>")
print ("<body>")
```

```
for i in range (12):
      print ("Super Tux %d<br>" % i )

print ("</body>")
print ("</html>")
```

permettra d'exécuter la commande le programme Python `ls` (liste des fichiers) sur le répertoire `/home`.
http://localhost/cgi-bin/listepy.cgi

16.3 PHP

Le PHP est un langage de programmation qui peut s'insérer dans le code HTML et qui sera transcrit par le serveur WEB au moment où la page Web est demandé.
Le code PHP doit être entouré de 2 balises spécifiques :

```
<?php
Code PHP
?>
```

Listing 16.4 – `firstpage.php`

```
<html>
<head>
<title>First page in PHP</title>
</head>
<body>
<?php
   print "coucou, je suis une page PHP";
?>
</body>
</html>
```

http://localhost/bd/firstpage.php

Nous allons maintenant adapter notre programme CGI en PHP, celui-ci devient :

Listing 16.5 – `secondpage.php`

```
<html>
<head>
<title>Seocnd page in PHP</title>
</head>
<body>
<?php
   for ($i=0; $i<12; $i++) {
      print "Super Tux $i<br>\n";
   }
?>
</body>
</html>
```

http://localhost/bd/secondpage.php

16.4 Les formulaires

16.4.1 Formulaire HTML

Nous allons commencer par créer un formulaire en HTML. Formulaire simple puisqu'on demandera simplement un mot (de 20 caractères maximum) et on le retransmettra.
Pour se faire, nous allons utiliser les mots clé HTML `FORM` et `INPUT` de la façon suivante :

Listing 16.6 – `formulaire_simple.html`

```
<HTML>
<HEAD>
<TITLE>Premier Formulaire</TITLE>
</HEAD>
<BODY>
<h1>Un premier formulaire</h1>
<FORM>
<INPUT TYPE="TEXT" SIZE=20 ></INPUT>
```

```
</FORM>
</BODY>
</HTML>
```

http://localhost/bd/formulaire_simple.html
L'affichage nous donne bien un champs de saisie mais ne permet pas de transmettre l'information. Pour cela nous devons ajouter une action qui sera réalisée lors du clic sur un bouton.

Listing 16.7 – formulaire_action.html

```
<HTML>
<HEAD>
<TITLE>Premier Formulaire</TITLE>
</HEAD>
<BODY>
<h1>Un premier formulaire</h1>
<FORM action="http://localhost/bd/formulaire_reponse.php" method="get">
<INPUT TYPE="TEXT" NAME="mot" SIZE=20></INPUT>
<INPUT TYPE="SUBMIT" VALUE="Envoyer l'information"></INPUT>
</FORM>
</BODY>
</HTML>
```

http://localhost/bd/formulaire_action.html

Observation

✎ On notera le champs NAME qui permet de différencier les différents champs du formulaire.

L'appui sur le bouton "Envoyer l'information" ouvre une page qui n'existe pas ce qui empêche tout traitement de cette dernière. Cependant, il est possible de remarquer dans l'url, la présence de `?mot=marvin` où `marvin` est la saisie que j'ai effectuée dans la zone de saisie.
Cette méthode d'envoi de l'information se nomme la méthode `GET`. Une autre méthode existe, la méthode `POST` qui n'affiche pas les données. En général, dans toute la phase de conception d'une page web, on préfère utiliser `GET` afin de contrôler le flux d'informations. Plus d'informations sur http://www.w3schools.com/tags/ref_httpmethods.asp.

16.4.2 Formulaire PHP

Nous allons utiliser le PHP pour récupérer les valeurs transmises dans l'url. Pour cela, la variable de type tableau associatif `$_GET`.

Listing 16.8 – formulaire_reponse.php

```
<HTML>
<HEAD>
<TITLE>Reponse Premier Formulaire</TITLE>
</HEAD>
<BODY>
<h1>Reponse premier formulaire</h1>

<?php
    if (!empty($_GET['mot'])){
       print "Le mot que vous avez saisi est ".$_GET['mot'];
   } else {
       print "Vous n'avez saisi aucun mot !";
   }
?>
</BODY>
</HTML>
```

Les deux programmes peuvent être combinés en un seul en PHP mais nous nous éloignons de nos bases de données ...

16.5 Connexion à une base de données en PHP

La connexion à une base de données se fait de manière similaire au Python, en voici un exemple :

Listing 16.9 – `connect_bd.php`

```
<html>
<head>
<title>Ma premiere connexion</title>
</head>
<body>
<?php

   $mysqli = new mysqli("localhost", "pizza", "pizzamdp", "FirstPizza");

   if ($mysqli->connect_errno) {
    echo "Failed to connect to MySQL: (" . $mysqli->connect_errno . ") " . $mysqli->connect_error;
   }

   $mysqli->real_query("SELECT nom_Pizza FROM Pizza ") or die ("Cannot fetch:". $mysqli->error."\n" );;
   $res = $mysqli->use_result();

   while ($row = $res->fetch_assoc()) {
      echo $row['nom_Pizza']."<br>";
   }

   $res->close();
?>
</body>
</html>
```

http://localhost/bd/connect_bd.php

16.6 Exercice

Écrire un programme web qui demande un login et un mot de passe puis indique à l'utilisateur si il a ou non le droit de se connecter. Pour se faire, suivre les étapes suivantes :

1. Créer une base de données `phpusers` sur laquelle l'utilisateur MySQL `phpusers / phpusers` dispose de l'ensemble des droits.
2. Sur cette base, créer une table `users` qui contient les informations suivantes :
   - Nom
   - Prénom
   - Courriel
   - Login
   - Mot de passe
3. Insérer des utilisateurs de votre choix dans la table en ligne de commande SQL
4. Écrire un programme PHP qui se connecte à votre base et qui lit les informations contenues dans cette dernière

5. Écrire une page Web contenant un formulaire de saisie du login et du mot de passe. On utilisera le type `PASSWORD` pour la zone de saisie du mot de passe.
   Pour connaître le nombre de résultats retournés par la requête on utilisera `mysqli_num_rows($res);`
6. 2 choix se présentent alors :
   — le couple login/motdepasse est correct, on affiche alors l'identité de l'utilisateur par un message de bienvenue.
   — le couple login/motdepasse est incorrect, on affiche alors un message indiquant une erreur dans la saisie.

16.6.1 Création de la base et de l'utilisateur associé

```
DROP DATABASE IF EXISTS phpusers;
CREATE DATABASE IF NOT EXISTS phpusers;
```

```
GRANT ALL ON `phpusers`.* to 'phpusers'@'localhost' identified by 'phpusers';
FLUSH PRIVILEGES;
```

16.6.2 Création de la table `users`

```
USE phpusers;

CREATE TABLE IF NOT EXISTS `users` (
  `id_users` int(11) NOT NULL AUTO_INCREMENT,
  `nom` varchar(50) NOT NULL,
  `prenom` varchar(30) NOT NULL,
  `courriel` varchar(50) NOT NULL,
  `login` varchar(50) NOT NULL,
  `mdp` varchar(50) NOT NULL,
  PRIMARY KEY (`id_users`)
) ENGINE=InnoDB;
```

16.6.3 Remplissage de la table `users`

```
USE phpusers;

INSERT INTO `users` (`id_users`, `nom`, `prenom`, `courriel`, `login`, `mdp`) VALUES
(1, 'kent', 'clark', 'clark.kent@superman.fr', 'clark', 'kent'),
(2, 'wayne', 'bruce', 'bruce.wayne@batman.fr', 'bruce', 'wayne'),
(3, 'berthomier', 'eric', 'ebrnospam@free.fr', 'eric', 'berthomier'),
(4, 'tux', 'tuxie', 'tux@linux.fr', 'tux', 'tuxie');
```

16.6.4 PHP - Lecture de la base

Listing 16.10 – `lecture_base.php`

```
<html>
<head>
<title>Lecture base users</title>
</head>
<body>
<?php

   $mysqli = new mysqli("localhost", "phpusers", "phpusers", "phpusers");

   if ($mysqli->connect_errno) {
    echo "Failed to connect to MySQL: (" . $mysqli->connect_errno . ") " . $mysqli->connect_error;
   }

   $mysqli->real_query("SELECT * FROM users") or die ("Cannot fetch:". $mysqli->error."\n" );;
   $res = $mysqli->store_result();

   while ($row = $res->fetch_assoc()) {
      foreach (array ('nom','prenom','courriel','login','mdp') as $valeur)
      {
         echo $valeur . " : ".$row[$valeur] . "<br>";
      }
      echo "<hr><br>";
   }
```

```
?>
</body>
</html>
```

16.6.5 HTML - Formulaire

Listing 16.11 – `formulaire.html`

```
<HTML>
<HEAD>
<TITLE>Formulaire Login / Mdp</TITLE>
</HEAD>
<BODY>
<h1>Formulaire Login / Mdp</h1>

<FORM action="http://localhost/bd/formulaire_login.php" method="get">

   <table>
   <tr>
      <td>Login : </td><td><INPUT TYPE="TEXT" NAME="login" SIZE=50></INPUT></td>
   </tr>
   <tr>
      <td>Mot de passe : </td><td><INPUT TYPE="PASSWORD" NAME="mdp" SIZE=50></INPUT></td>
   </tr>
   </table>
   <INPUT TYPE="SUBMIT" VALUE="Se connecter"></INPUT><br>

</FORM>

</BODY>
</HTML>
```

http://localhost/bd/formulaire.html

16.6.6 PHP - Validation Login / Mdp

Listing 16.12 – `formulaire_login.php`

```
<HTML>
<HEAD>
<TITLE>Login</TITLE>
</HEAD>
<BODY>
   <h1>Login</h1>

<?php

   $fin = 0;

   if (!empty($_GET['login'])){
      print "Le login que vous avez saisi est ".$_GET['login']."<br>";
      $login = $_GET['login'];
   } else {
      print "Vous n'avez saisi aucun login !"."<br>";
      $fin=1;
   }
    if (!empty($_GET['mdp'])){
      print "Le mot de passe que vous avez saisi est ".$_GET['mdp']."<br>";
      $mdp = $_GET['mdp'];
   } else {
      print "Vous n'avez saisi aucun mot de passe !"."<br>";
      $fin=1;
   }

   if ($fin)
   {
      print "Saisie incorrecte";
   } else {
      print "Recherche de ". $login . " / ". $mdp. "<br>";

      $mysqli = new mysqli("localhost", "phpusers", "phpusers", "phpusers");

      if ($mysqli->connect_errno) {
       echo "Failed to connect to MySQL: (" . $mysqli->connect_errno . ") " . $mysqli->connect_error;
```

```
      }

      $mysqli->real_query("SELECT * FROM users WHERE login=\"".$login."\" AND mdp=\"".$mdp."\"") or die ("Cannot
           fetch:". $mysqli->error."\n");
      $res = $mysqli->store_result();

      print "Nombre de resultats : ".mysqli_num_rows($res)."<br>";
      if (mysqli_num_rows($res) == 1 )
      {
         echo "<br><hr><br>";
         $row = $res->fetch_assoc();
         foreach (array ('nom','prenom','courriel','login','mdp') as $valeur)
         {
            echo $valeur . " : ".$row[$valeur] . "<br>";
         }
         echo "<br><hr><br>";
      } else {
         echo "<br><hr><br>";
         print "Vous ne pouvez vous connecter.<br>";
         echo "<br><hr><br>";
      }
   }
?>

</BODY>
</HTML>
```

16.7 Sécurité : SQLInjection

Lors de la saisie du mot de passe, testez les logins / mots de passe suivants :

— `1" or "1" = "1`

— `1' or '1' = '1`

La requête devient `SELECT * FROM users WHERE login = "clark" AND mdp = "1" OR "1" = "1"` ce qui permet de rendre toujours vraie cette dernière.

16.7.1 SQLInjection : À vous de jouer ...

Corentin un programmeur a utilisé la requête suivante pour valider que le mot de passe d'un utilisateur était correct :

```
SELECT id, nom FROM users WHERE nom="$user" AND motdepasse="$mdp";
```

Si la requête retourne un résultat alors le couple login / mot de passe est correct, sinon, il ne l'est pas. Trouver comment avoir toujours un bon mot de passe à l'aide du caractère de commentaire de MySQL.

```
mysql> select * from users;
+----+---------+---------------------+
| id | nom | motdepasse |
+----+---------+---------------------+
| 9 | admin | truc |
```

```
| 10 | alban | tbeau |
| 11 | simon | cussoinait |
| 12 | bonjour | ceciestunmotdepasse |
+----+---------+---------------------+
```

Il suffit de placer un commentaire dans le nom d'utilisateur pour annulé le reste de la requête.

```
mysql> SELECT id, nom FROM users WHERE nom="admin";#" AND motdepasse="toto";
+----+-------+
| id | nom |
+----+-------+
| 9 | admin |
+----+-------+
1 row in set (0,00 sec)
```

16.8 Retour sur nos pizzas

Nous l'avions indiqué au début de ce cours, le but final de l'application est de pouvoir gérer la commande des pizzas et du stock. Nous allons créer une interface Web permettant de choisir une pizza, parmi celles proposées.

16.8.1 HTML

Une liste de choix simple se fait en HTML de la façon suivante :

Listing 16.13 – `choix.html`

```
<html>
<head>
   <title>Liste de choix</title>
</head>
<body>
   <FORM ACTION="http://localhost">
      <SELECT NAME="liste_choix">
         <OPTION VALUE="1">Valeur 1</OPTION>
         <OPTION VALUE="2">Valeur 2</OPTION>
         <OPTION VALUE="3">Valeur 3</OPTION>
         <OPTION VALUE="4">Valeur 4</OPTION>
         <OPTION VALUE="5">Valeur 5</OPTION>
      </SELECT>
      <INPUT TYPE="submit" VALUE="Choisir">
   </FORM>
</body>
</html>
```

`http://localhost/bd/choix.html`

16.8.2 Exercice

Modifier le programme HTML en programme PHP afin de pouvoir choisir la pizza de son choix dans la base `FirstPizza`. Une fois la pizza choisie par l'utilisateur, l'ensemble des informations liées à cette dernière sont trasmises à l'usager via la page Web.

Figure 16.4 – Exemple d'exécution du programme

Listing 16.14 – `choix_pizza.php`

```
<html>
<head>
   <title>Liste de choix</title>
</head>
<body>
<?php

   $mysqli = new mysqli("localhost", "pizza", "pizzamdp", "FirstPizza");

   if ($mysqli->connect_errno) {
    echo "Failed to connect to MySQL: (" . $mysqli->connect_errno . ") " . $mysqli->connect_error;
   }

   $mysqli->real_query("SELECT id_Pizza, nom_Pizza FROM Pizza ") or die ("Cannot fetch:". $mysqli->error."\n" );;
   $res = $mysqli->use_result();

   print ('<h1>Pizza au choix</h1>');
   print ('<FORM ACTION="http://localhost/bd/affiche_pizza.php">');
   print ('<SELECT NAME="liste_choix">');

   while ($row = $res->fetch_assoc()) {
      echo '<OPTION VALUE="'.$row['id_Pizza'].'">'.$row['nom_Pizza'].'</OPTION>';
   }

   print ('</SELECT>');
   print ('<INPUT TYPE="submit" VALUE="Choisir">');
   print ('</FORM>');

   $res->close();
?>

</body>
</html>
```

Listing 16.15 – `affiche_pizza.php`

```
<html>
<head>

<?php

   if ( ! isset($_GET['liste_choix'] )) {
      print <<< FIN
<title>Aucune pizza choisie</title>
</head>
<body>
Aucune pizza.<br>
<a href="http://localhost/bd/choix_pizza.php">Choisir une pizza ...</a>

FIN
;
   } else {

      $choix = $_GET['liste_choix'];

      $mysqli = new mysqli("localhost", "pizza", "pizzamdp", "FirstPizza");
      $mysqli -> autocommit (TRUE);

      if ($mysqli->connect_errno) {
         echo "Failed to connect to MySQL: (" . $mysqli->connect_errno . ") " . $mysqli->connect_error;
      }

      $requete = "SELECT * FROM Pizza WHERE id_Pizza=".$choix;

      $mysqli->real_query($requete) or die ("Cannot fetch:". $mysqli->error."\n");;
      $res = $mysqli->use_result();
      $row = $res->fetch_assoc();

      $pizza = $row;
      $res-> close();

      print "<title>Pizza ".$pizza['nom_Pizza']."</title>";
```

```
        print "</head>";
        print "<body>";

        print "<center><h1>Pizza ".$pizza['nom_Pizza']."</h1></center>";

        print "<br>";
        print "<ul>";

        $requete = "SELECT nom_pate FROM Pate WHERE id_Pate=".$pizza['fk_id_Pate'];
        //~ print "$requete";

        $mysqli->real_query($requete) or die ("Cannot fetch:". $mysqli->error."\n");;
        $res = $mysqli->use_result();

        $row = $res->fetch_assoc();
        print "<li>Pate : ".$row['nom_pate']."</li>";
        $res-> close();

        $requete = "SELECT nom_garniture FROM Garniture WHERE ( id_Garniture IN (SELECT `fk_id_Garniture` FROM `
             GarniturePizza` WHERE `fk_id_Pizza`=".$choix."))";
        //~ print "$requete";

        $mysqli->real_query($requete) or die ("Cannot fetch:". $mysqli->error."\n");;
        $res = $mysqli->use_result();

        while ($row = $res->fetch_assoc()) {
           print "<li>Garniture : ".$row['nom_garniture']."</li>";
        }

        print "</ul>";
        $res-> close();

    }
?>

</body>
</html>
```

http://localhost/bd/choix_pizza.php

16.9 Formulaires rapides

Le site http://wwww.phpform.org vous permet de créer vos formulaires dynamiquement pour plus de rapidité.

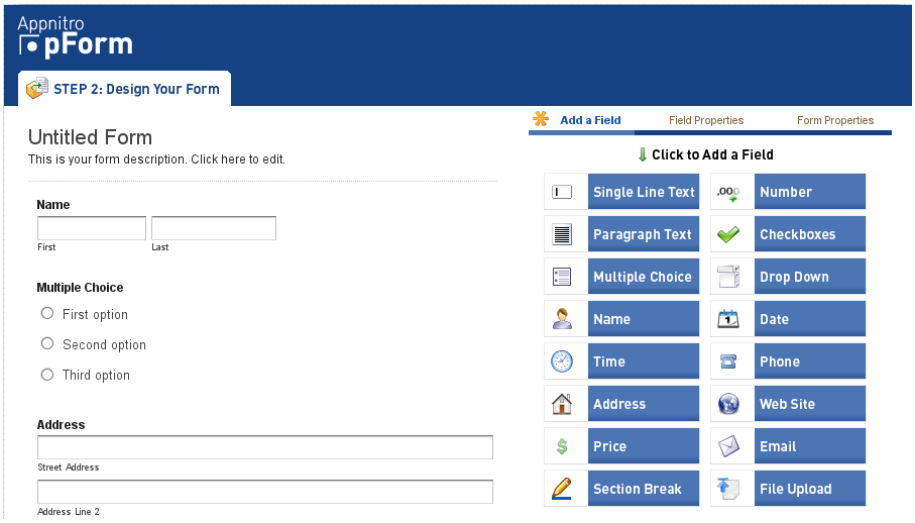


FIGURE 16.5 – phpForm

Sixième partie

Annexes

Chapitre 17

Les Accents

Nous avons supprimé les accents des éléments contenus dans les différentes tables. En fait, leur gestion est toujours complexe pour un non initié.

Voici, néanmoins un exemple de programme qui permet d'insérer les données dans une table avec les accents et ceci dans n'importe quel environnement.
Pour cela, il est nécessaire de passer tout en UTF8 en commençant par la création de la table.

17.1 La table

Listing 17.1 – `affiche_pizza.php`

```
mysql> create table uGarniture (uNom_Garniture VARCHAR (50)) CHARACTER SET utf8;
```

17.2 Le fichier d'entrée

L'ensemble des données doit être lui aussi en UTF8.
Nous utiliserons ici un fichier nommé `garniture.txt`, dans le cas d'une saisie, on veillera soit à la tranformer soit à faire la saisie en UTF8.

- Gruyère
- Crème Fraiche
- Fromage de chèvre frais
- Porc Braisé
- ...

17.3 Le programme

Le programme doit lui aussi être adapté :

Listing 17.2 – `corr_garniture_unicode.py`

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
import MySQLdb as mdb
import sys

conn = mdb.connect( host="localhost",  # votre hôte, normalement le serveur local
                user="pizza",   # utilisateur
                passwd="pizzamdp",  # mot de passe
                db="FirstPizza", # nom de la base de données
                charset="utf8")  # encodage des requêtes

# Traitement des requêtes aussitôt
conn.autocommit(True)
```

```
# Il est nécessaire de créer un objet de type Curseur pour réaliser des requêtes
curs = conn.cursor()

# On ouvre le fichier "garniture.txt"
f = open('garniture.txt', 'r')


for line in f:
    requete = 'insert into uGarniture set uNom_Garniture="' + line.rstrip() + '"'

    try:
        curs.execute(requete)

    except mdb.Error, e:
        print "Erreur %d: %s" % (e.args[0],e.args[1])
        sys.exit(1)

curs.close()
conn.close()

f.close ()
```

17.4 Le final

PhpMyAdmin nous permet d'avoir une vue des différentes données très rapidement. Les accents ont été conservés.

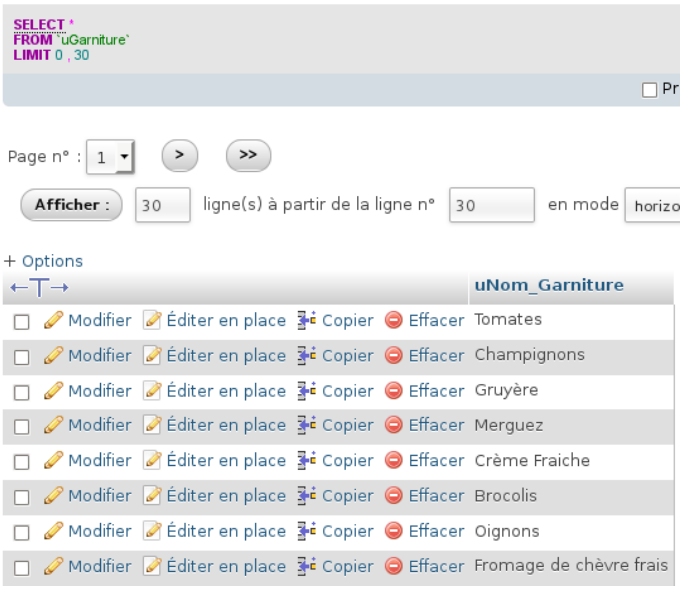


FIGURE 17.1 – Accents - UTF8 - PhpMyAdmin

Chapitre 18

phpMyAdmin

18.1 Introduction

PhpMyAdmin est véritablement le panneau d'administration standard d'une base de données. Il est utilisé pratiquement partout et même avec les hébergements ne proposant pas de CPanel.

C'est en réalité un ensemble de pages PHP (et en rien un programme) qui simplifient la tâche du programmeur en offrant une interface simple et efficace pour gérer les différentes bases de données du site.

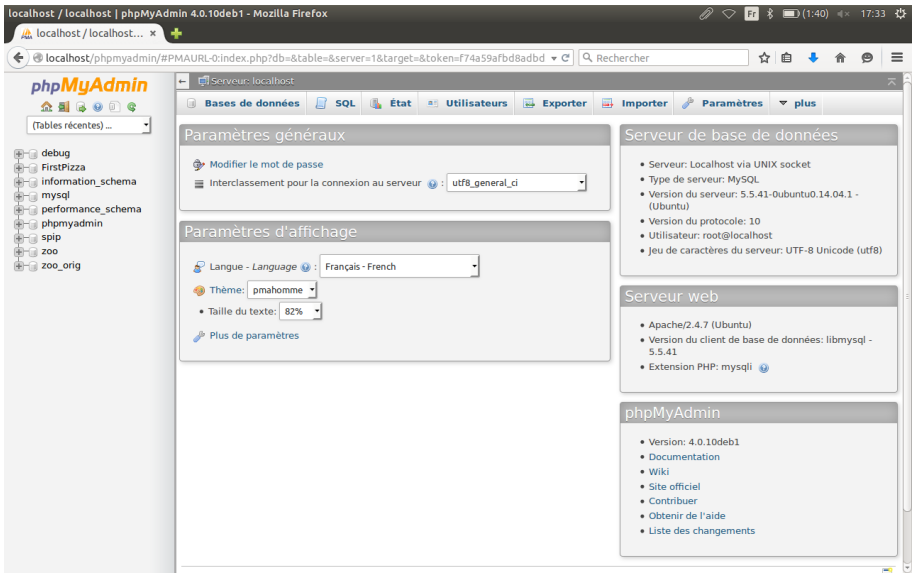


Figure 18.1 – PhpMyAdmin - Présentation générale

18.2 Menu de gauche

La partie à gauche représente les bases de données et notamment les bases `information_schema`, `mysql` et suivant les cas de figure `phpmyadmin`. La base de données `phpmyadmin` est liée à l'utilisation du logiciel du même nom.

FIGURE 18.2 – PhpMyAdmin - Menu de gauche

Dans l'ordre de gauche à droite :

— La petite maison en haut vous permettra de retourner sur cette page d'accueil.
— La porte vous permettra de clôturer votre session (très utile pour tester les droits d'un utilisateur).
— Le disque dur vous permettra de rentrer dans le gestionnaire de requêtes.
— Le ? vous affichera des manuels d'utilisation (en anglais) de phpMyAdmin
— Le feuillet avec un disque pointe sur la documentation MySQL
— La flèche courbe verte permet de rafraichir l'affichage

18.3 Menu principal

FIGURE 18.3 – PhpMyAdmin - Menu principal

Le menu principal vous permet d'accéder à des informations liées au serveur.

— Bases de données vous permet de gérer les bases de données de votre serveur.
— SQL vous permet de disposer d'une interface de requêtes SQL
— État vous permet d'accéder aux statistiques d'utilisation du serveur
— Utilisateurs vous permet de géréer vos utilisateurs
— Exporter / Importer vous permet de gérer la sauvegarde de vos bases.
— ...

Comme vous pouvez le constater, l'utilisation de phpMyAdmin est très intuitive, quelques astuces cependant.

18.4 Astuces

18.4.1 Utilisateurs

Lorsque vous créez un utilisateur, il est possible de lui créer une base de données associée dans le même temps, gain de temps et de sécurité (on a fait le minimum).

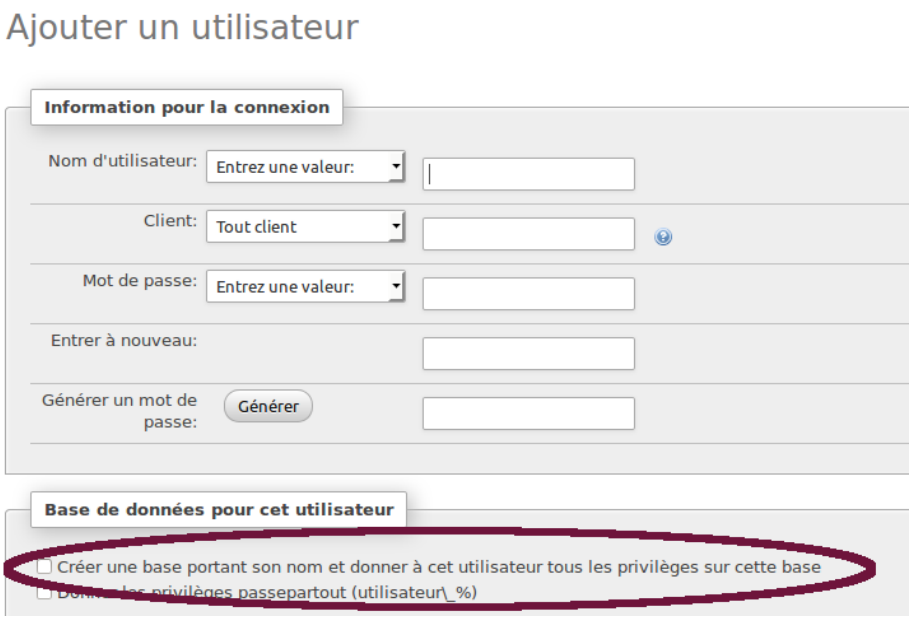


FIGURE 18.4 – PhpMyAdmin - Gestion des utilisateurs

A noter qu'il est possible de faire cette manipulation même si la base de données a déjà été créée (pas de création de cette dernière mais simplement une affectation des droits).

18.4.2 Renommage

Renommer une base de données peut conduire à de nombreuses erreurs humaines, phpMyAdmin le fait pour vous.

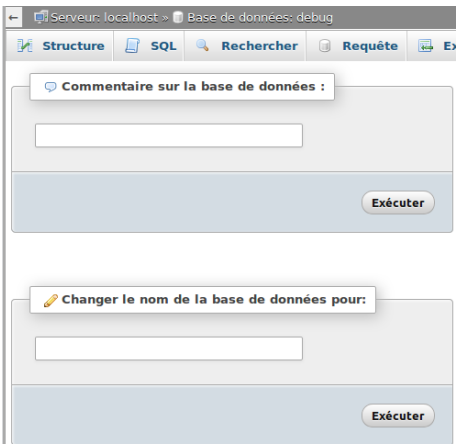


FIGURE 18.5 – PhpMyAdmin - Renommage d'une base

18.4.3 Export

Il est très souvent demandé de réaliser des exports des bases afin de pouvoir créer des documents hors ligne (tableau ...). PhpMyAdmin vous facilite une nouvelle fois la vie en vous permettant de réaliser des exports sous des formes très variées.

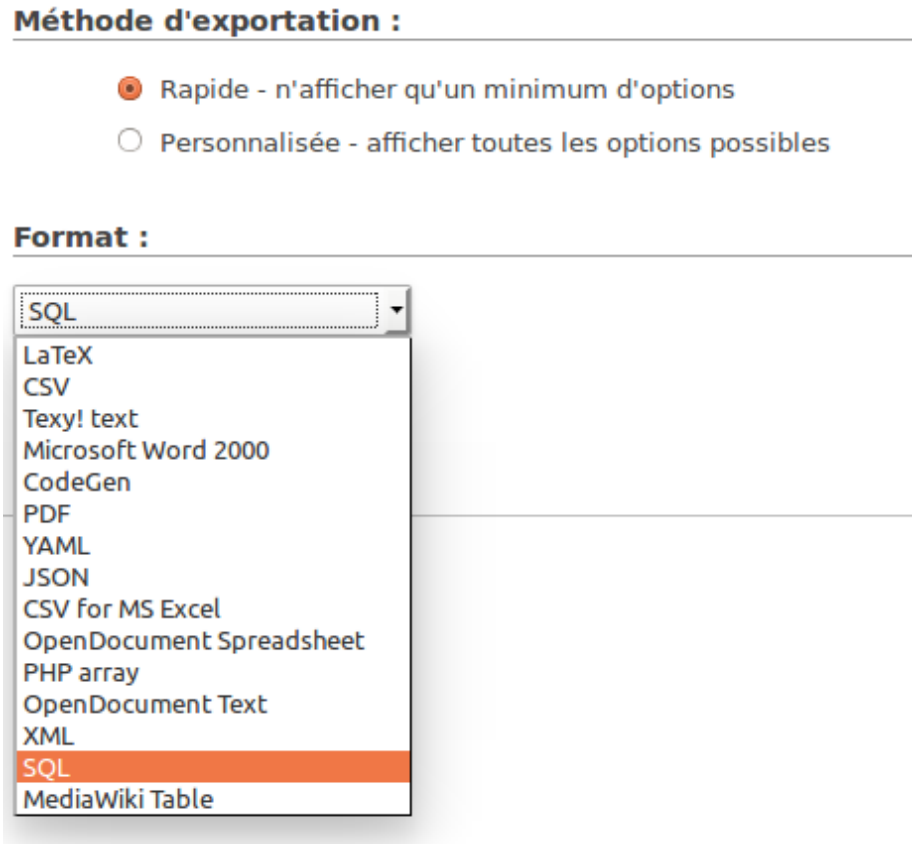


FIGURE 18.6 – PhpMyAdmin - Export d'une table

Remarque

✓ Il est possible de réaliser un export d'un résultats. Pour cela exécuter la commande SQL puis cliquer sur Export dans le bas de la page de résultats

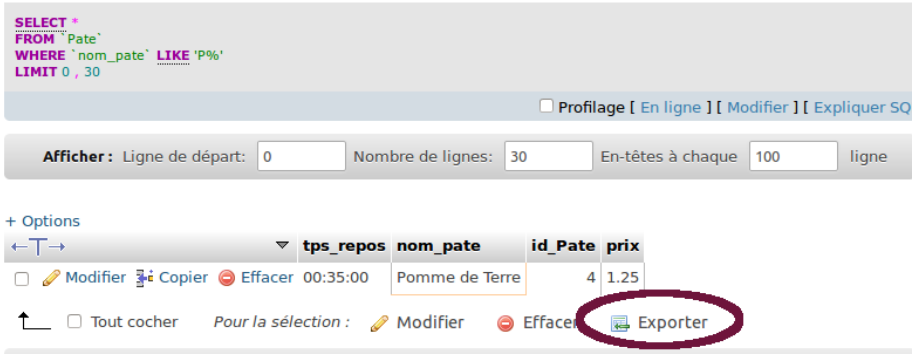


FIGURE 18.7 – PhpMyAdmin - Export des résultats d'une requête

18.4.4 Sauvegarde

Á l'image de l'export, phpMyAdmin vous propose de sauvegarder vos bases de données. Une astuce consiste à passer en mode `Personnalisée` et ainsi obtenir des options intéressantes.
Parmi celles-ci `Ajouter un énoncé DROP DATABASE` qui vous permettra de réinjecter votre base de données directement sans supprimer l'ancienne (attention à ce que vous faites tout de même !)

18.4.5 Concepteur

Le concepteur vous permet de modéliser votre base de données sous forme graphique et vous donne ainsi accès à un joli schéma.

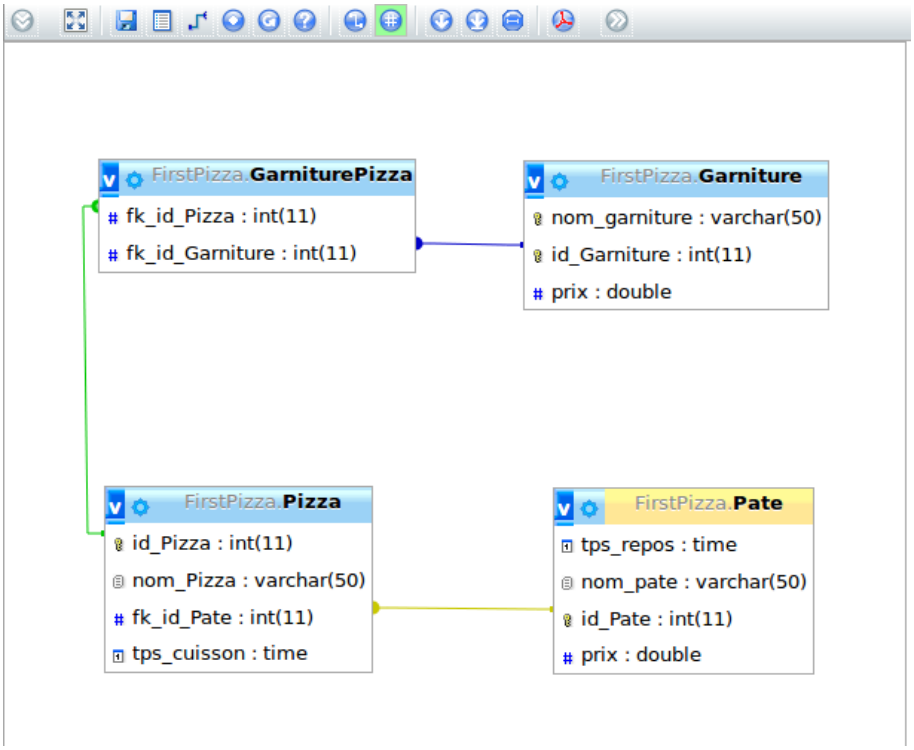


FIGURE 18.8 – PhpMyAdmin - Concepteur de Bases de Donnnées

Chapitre 19

MySQL Workbench

19.1 Description - Extrait du site

Source : http://dev.mysql.com/downloads/workbench/
MySQL Workbench est un outil visuel complet à destination des architectes de bases de données, des développeurs et ades admininistrateurs de Bases de Données (DBA). MySQL Workbench permet la modélisation de données, le développement SQL et intègre des outils d'administration compréhensifs pour la configuration des serveurs, l'administration des utilisateurs, les sauvegardes et bien plus encore. MySQL Workbench est disponible sous Windows, Linux et Mac OS X.

19.2 Installation

L'installation de MySQLWorkbench se fait en téléchargeant les outils sur URL, ou en tapant la commande suivant sous Debian / Ubuntu :

Listing 19.1 – `corr_garniture_unicode.py`

```
apt-get install mysql-workbench
```

19.3 Mode opératoire standard

Créer une connexion à la base de données que vous souhaitez analyser.

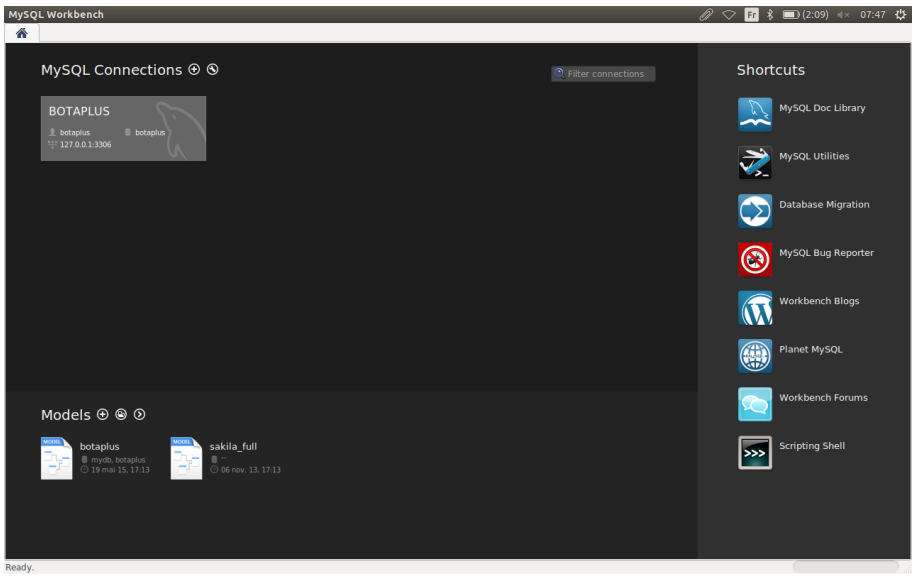


Figure 19.1 – MySQL-Workbench - Création d'une connexion

Saisir l'ensemble des valeurs nécessaires puis tester la connexion.

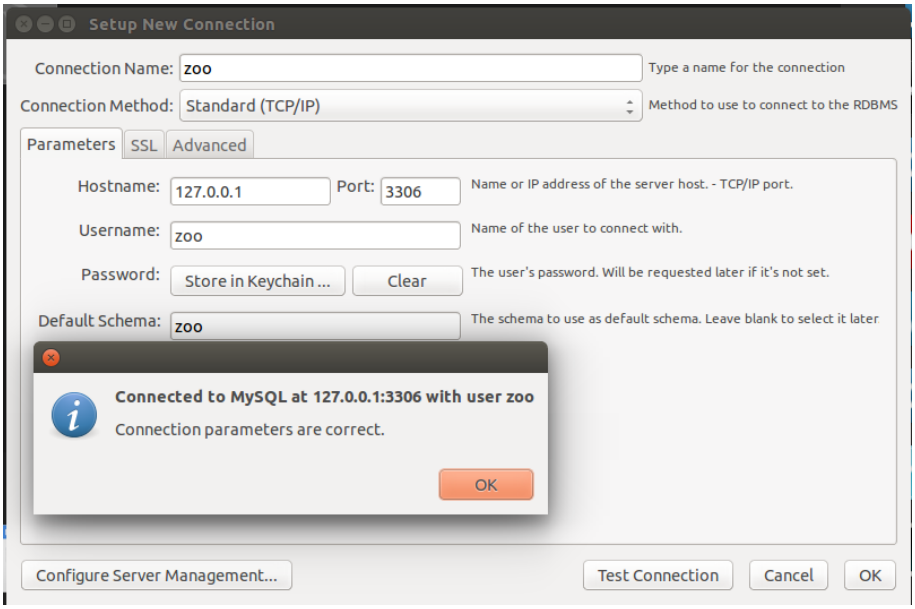


Figure 19.2 – MySQL-Workbench - Renseignement d'une connexion

Double cliquer sur la connexion ainsi réalisée, indiquer le mot de passe, vous êtes maintenant connecté à votre base de données.

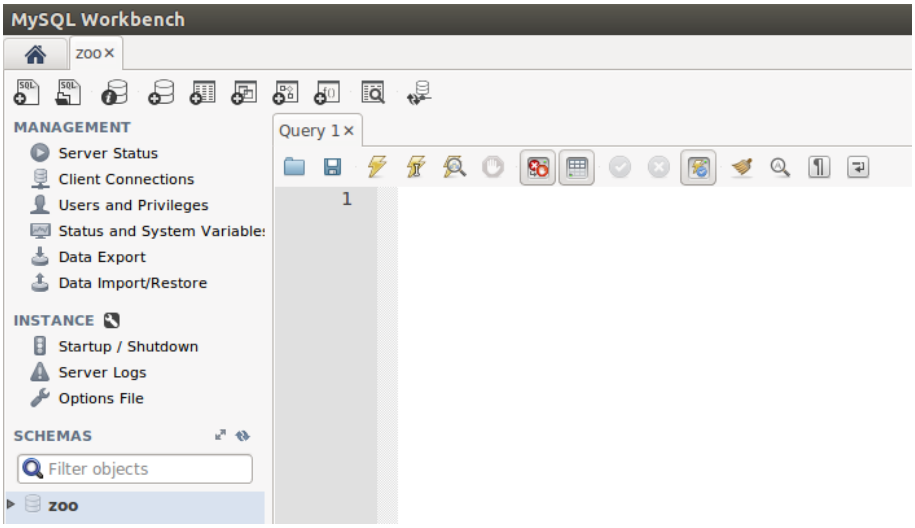


Figure 19.3 – MySQL-Workbench - Environnement graphique

19.4 Ingénierie inverse

Le principe de l'ingénierie inverse est de définir la conception d'un logiciel, ou d'une base de données dans notre cas, à partir du livrable. Imaginons, par exemple qu'une société souhaite reprendre le logiciel développé pour la gestion des Pizzas, elle va devoir reconstruire les relations établies dans la base de données à la main !

Pour disposer d'une réprésentation graphique des relations existantes dans une base de données, il faut se rendre dans le menu Database puis choisir Reverse Engineer ou appuyer simultanément sur les touches Ctrl + R. Une fois les informations nécessaires à la connexion transmises, une représentation graphique de la base apparaît.

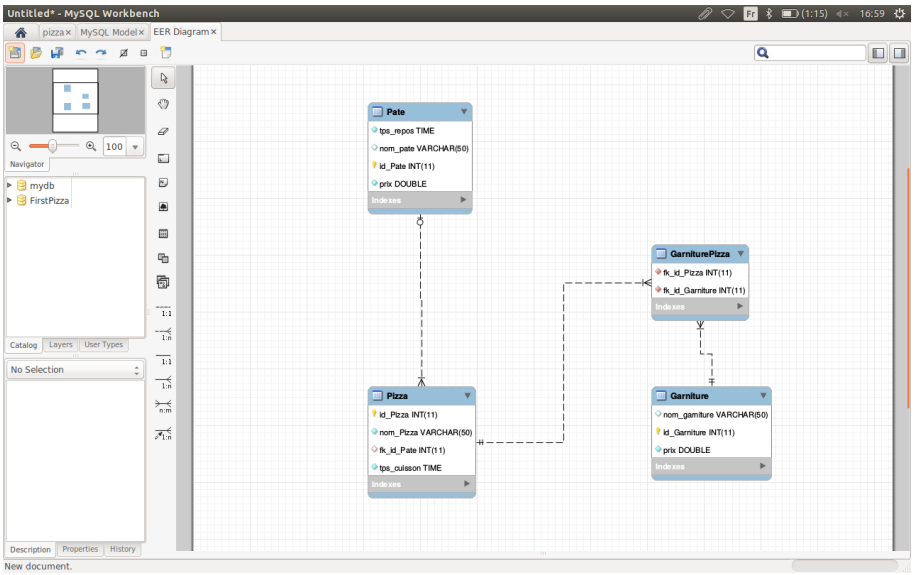


Figure 19.4 – MySQL-Workbench - Ingénierie Inverse

Liste des tableaux

3.1 Table `Pizza` . . . 23
3.2 Caractéristiques de la table `Pizza` . . . 25

4.1 Temps de cuisson . . . 33
4.2 Temps de repos . . . 34

6.1 Prix des Garnitures (€) . . . 42
6.2 Prix des Pâtes . . . 43

8.1 Cours - Typage en MySQL . . . 62

10.1 Contenu de la table `Animaux` . . . 71
10.2 Contenu de la table `Cage` . . . 71
10.3 Contenu de la table `Responsables` . . . 71
10.4 Contenu de la table `Maladies` . . . 71
10.5 Contenu de la table `Employes` . . . 72
10.6 Contenu de la table `Gardiens` . . . 72
10.7 Zoo - Clés primaires . . . 72
10.8 Zoo - Typage des attributs . . . 72

14.1 Structure de la table panier . . . 93
14.2 Contenu de la table panier . . . 93
14.2 Contenu de la table panier (suite) . . . 94
14.3 Structure de la table articles . . . 94
14.4 Contenu de la table articles . . . 94
14.5 Structure de la table traces . . . 94
14.5 Structure de la table traces (suite) . . . 95

Table des figures

3.1 Schéma de la base de données Pizza . . . 31

5.1 Source Sugar Mama . . . 39

6.1 Exemple pour la compréhension des jointures . . . 46

8.1 Schéma Base de Données - Pizzas - Cours . . . 56
8.2 Clés primaires et étrangères - Pizzas - Cours . . . 57
8.3 Jointure - Pizzas - Cours . . . 59
8.4 Schéma Base de Données - Pizzas - Suppression . . . 61

9.1 Zoo - Exemple de Schéma Relationnel applicable au problème . . . 70

10.1 Zoo - Schéma relationnel . . . 73

12.1 Cours - Sécurité - Table User . . . 86
12.2 Cours - Sécurité - Droits utilisateurs . . . 87

14.1 Triggers - Base de données Exemple . . . 93
14.2 Triggers - Base de données Exemple 2 . . . 95

16.1 Communication Client Serveur WEB (1/2) . . . 104
16.2 Communication Client Serveur WEB (2/2) . . . 104
16.3 Communication Client Serveur WEB LAMP/WAMP . . . 105
16.4 Exemple d'exécution du programme . . . 112
16.5 phpForm . . . 114

17.1 Accents - UTF8 - PhpMyAdmin . . . 118

18.1 PhpMyAdmin - Présentation générale . . . 119
18.2 PhpMyAdmin - Menu de gauche . . . 120
18.3 PhpMyAdmin - Menu principal . . . 120
18.4 PhpMyAdmin - Gestion des utilisateurs . . . 121
18.5 PhpMyAdmin - Renommage d'une base . . . 121
18.6 PhpMyAdmin - Export d'une table . . . 122
18.7 PhpMyAdmin - Export des résultats d'une requête . . . 122
18.8 PhpMyAdmin - Concepteur de Bases de Donnnées . . . 123

19.1 MySQL-Workbench - Création d'une connexion . . . 125
19.2 MySQL-Workbench - Renseignement d'une connexion . . . 126
19.3 MySQL-Workbench - Environnement graphique . . . 126
19.4 MySQL-Workbench - Ingénierie Inverse . . . 127