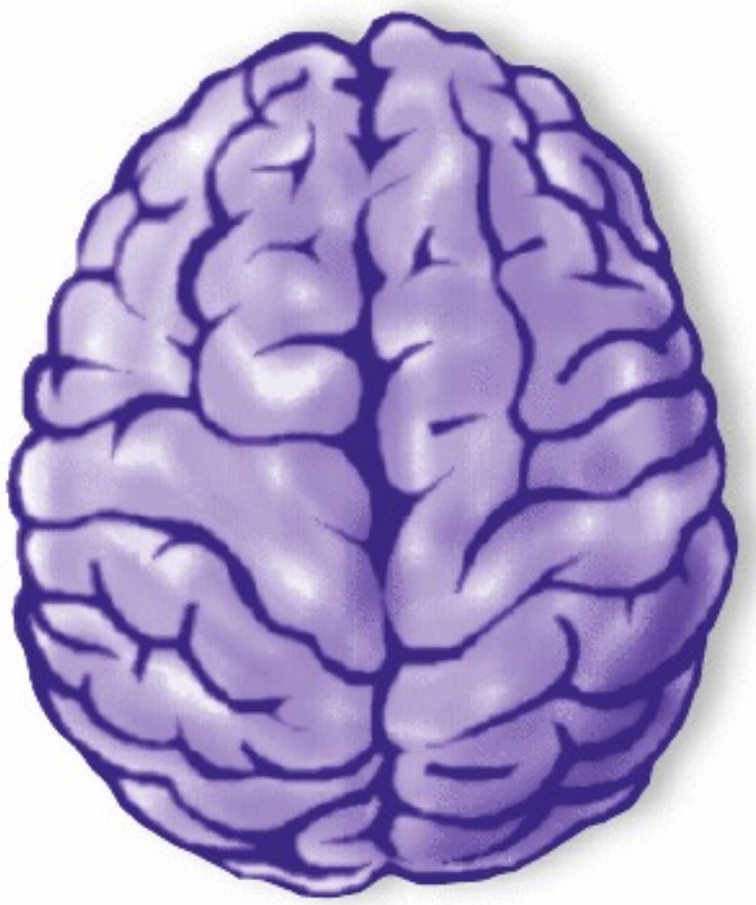
Un guide d'apprentissage

SQL Tête la première



Lynn Beighley
Traduction de Sophie Govaere

DIGIT BOOKS

SQL

Tête la première

Les bases de données chez Digit Books

SQL en concentré, 2è édition

SQL par l'exemple

SQL à 200%

SQL — précis et concis

LINQ, l'avenir de l'accès aux données avec C# 3.0

Maîtriser MySQL 5

MySQL — précis & concis

PostgreSQL par la pratique

La collection *Tête la première*

Analyse et conception orientées objet — Tête la première

Ajax — Tête la première

HTML avec CSS et XHTML — Tête la première

Design Patterns — Tête la première

Java™ — Tête la première

C# — Tête la première (2008)

SQL Tête la première

Lynn Beighley

Traduction de Sophie Govaere

SQL — Tête la première

de Lynn Beighley

© 2007 O'Reilly Media, Inc.

© 2009 Digit Books, pour la traduction française

ISBN : 978-2-8150-0002-4

Prix : 37 euros

Conception : *Kathy Sierra, Bert Bates*

Responsable éditorial VO : *Brett D. McLaughlin*

Éditeur VO : *Catherine Nolan*

Éditrice VF : *Dominique Buraud*

Design : *Louise Barr*

Couverture conçue par : *Louise Barr, Karen Montgomery*

Production : *Sanders Kleinfeld*

Index : *Julie Hawks*

Relecture VO : *Andrew Fader*

Relecture VF : *Olivier Fillol*

Il est incroyablement patient.

Les programmes figurant dans ce livre ont pour but d'illustrer les sujets traités. Il n'est donné aucune garantie quant à leur fonctionnement une fois compilés, assemblés ou interprétés dans le cadre d'une utilisation professionnelle ou commerciale.

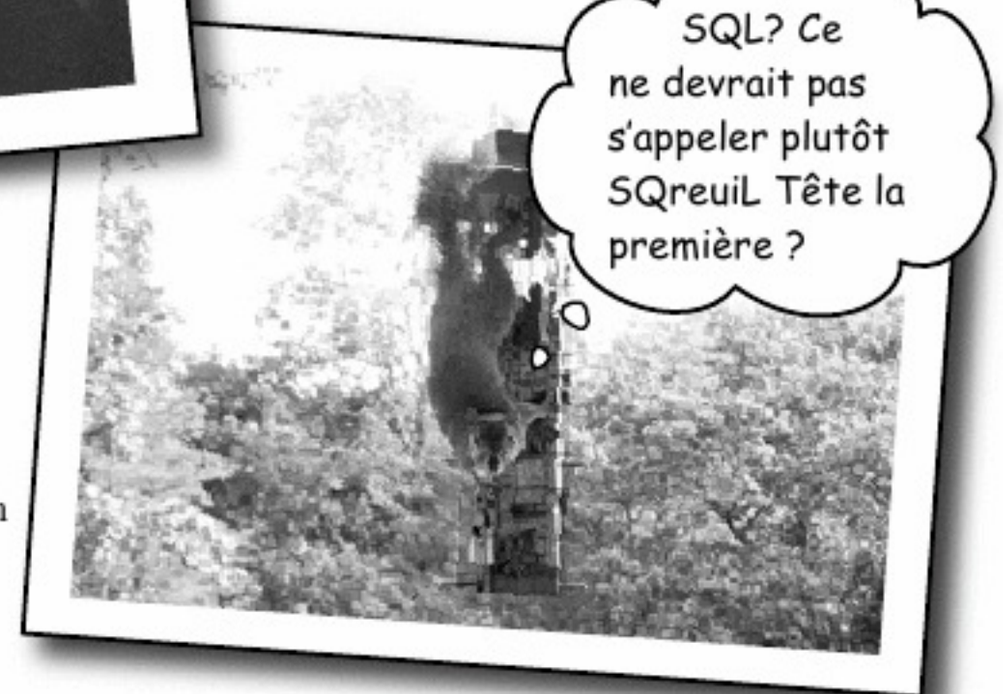
Toute représentation ou reproduction, intégrale ou partielle, faite sans le consentement de l'auteur, de ses ayants droit, ou ayants cause, est illicite (loi du 11 mars 1957, alinéa 1er de l'article 40). Cette représentation ou reproduction, par quelque procédé que ce soit, constituerait une contrefaçon sanctionnée par les articles 425 et suivants du Code pénal. La loi du 11 mars 1957 autorise uniquement, aux termes des alinéas 2 et 3 de l'article 41, les copies ou reproductions strictement réservées à l'usage privé du copiste et non destinées à une utilisation collective d'une part et, d'autre part, les analyses et les courtes citations dans un but d'exemple et d'illustration.

Aucun clown, crêpe ni Jeannette n'ont été blessé pendant la réalisation de ce livre. Seulement ma voiture, mais elle a été réparée.

À notre monde, inondé de données.
Et à vous, qui voulez les maîtriser.

Auteur de SQL — Tête la première

Lynn Beighley



Vue de la fenêtre de Lynn.

Lynn est un écrivain de fiction coincée dans le corps d'un écrivain technique. En découvrant que l'écriture de livres techniques rapportait en fait de l'argent concret, elle a appris à l'accepter et à l'apprécier.

Après un retour à l'école pour obtenir un Master en informatique, elle a travaillé pour les acronymes NRL et LANL. Puis elle a découvert Flash et a écrit son premier bestseller.

Victime des circonstances, elle a déménagé dans la Silicon Valley juste avant le grand krach. Elle a passé plusieurs années à travailler pour Yahoo! et à écrire d'autres livres et des formations. S'adonnant enfin à son penchant pour l'écriture créative, elle a déménagé dans la région de New York pour passer un Master en écriture créative.

Sa thèse dans le style Tête la première a été entendue par une salle pleine de professeurs et d'étudiants. Elle a été extrêmement bien reçue et Lynn a obtenu son diplôme, terminé *SQL Tête la première* et attend impatiemment de commencer son prochain livre.

Lynn adore voyager, cuisiner et inventer des histoires élaborées sur le passé de personnes inconnues. Elle a un peu peur des clowns.

Table des matières (version courte)

	Intro	xxv
1	Données et tables : *Chaque chose à sa place*	1
2	L'instruction SELECT : *Un talent pour la récupération de données*	53
3	DELETE et UPDATE : *Le changement vous fera du bien*	119
4	Conception intelligente de table : *À quoi bon être normal ?*	159
5	ALTER : *Réécrire le passé*	197
6	SELECT avancé : *Un regard neuf sur vos données*	235
7	Conception de bases de données multi-tables : *Dépasser les limites de votre table*	281
8	Jointures et opérations multi-tables : *Est-ce qu'on ne pourrait pas s'entendre ?*	343
9	Sous-requêtes : *Des requêtes dans les requêtes*	379
10	Jointures externes, auto-jointures et unions : *Nouvelles tactiques*	417
11	Contraintes, vues et transactions : *Trop de cuisiniers gâtent la base de données*	455
12	Sécurité : *Protéger vos biens*	493

Table des matières (version longue)

Intro

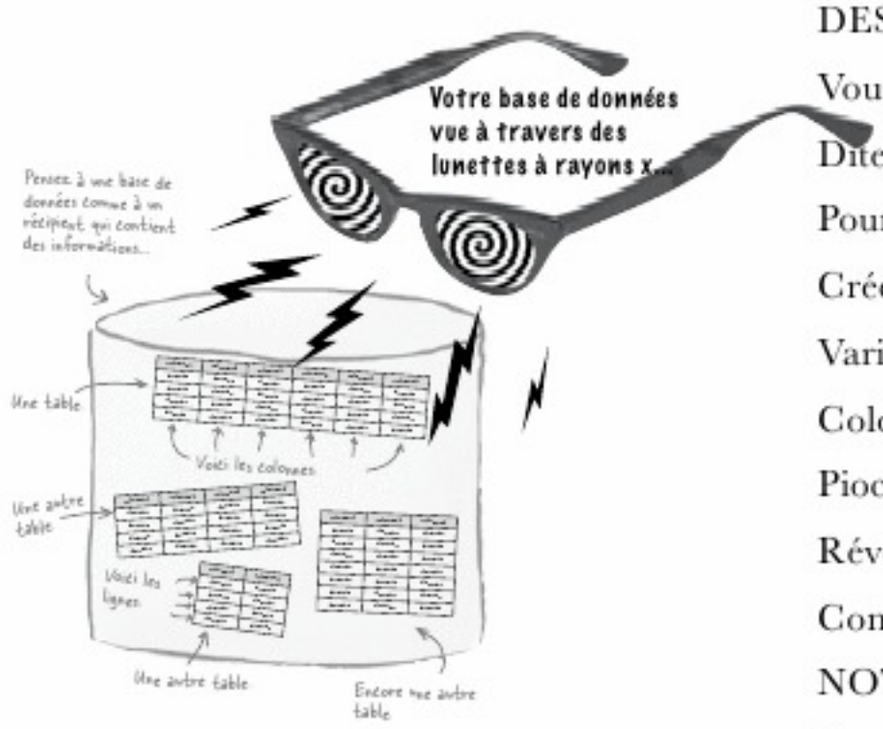
Votre cerveau et SQL. Voilà que vous essayez d'*apprendre* quelque chose et que votre *cerveau* fait tout ce qu'il peut pour vous empêcher de le *mémoriser*. Votre cerveau pense : « Mieux vaut laisser de la place pour les choses vraiment importantes, comme savoir quels sont les animaux sauvages qu'il vaut mieux éviter ou que faire du snowboard en short n'est pas une bonne idée. ». Mais comment procéder pour lui faire croire que votre vie dépend de vos connaissances en SQL ?

À qui s'adresse ce livre ?	xxvi
Nous savons ce que vous pensez	xxvii
La métacognition	xxix
Domptez votre cerveau	xxxi
Lisez-moi	xxxii
L'équipe technique	xxxiv
Remerciements	xxxv

données et tables

1 Chaque chose à sa place

Vous ne trouvez pas énervant de perdre quelque chose ? Que ce soient vos clés de voiture, ce bon de réduction de 25% chez Ikéa ou les données de votre application, il n'y a rien de pire que de ne pas pouvoir **trouver ce dont on a besoin**... au moment où on en a besoin. Pour vos applications, il n'existe pas de meilleur endroit qu'une **table** pour conserver vos informations importantes. Alors tournez la page, entrez et venez vous promener dans le monde des **bases de données relationnelles**.



Définir vos données	2
Regardez vos données triées par catégories	7
Qu'y a-t-il dans une base de données ?	8
Votre base de données vue à travers des lunettes à rayons x...	10
Les bases de données contiennent des données reliées entre elles	12
Les tables vues de plus près	13
Prenez les commandes !	17
Dresser la table : l'instruction CREATE TABLE	19
Créer une table plus compliquée	20
Voyez comme SQL est facile à écrire	21
Créer la table mes_contacts, enfin	22
Votre table est prête	23
Une réunion avec quelques types de données	24
DESCription de votre table	28
Vous ne pouvez pas recréer une table ou une base de données existante !	30
Dites au revoir à la vieille table et bonjour à la nouvelle	32
Pour ajouter des données dans votre table, vous utilisez l'instruction INSERT	34
Créer l'instruction INSERT	37
Variations sur une instruction INSERT	41
Colonnes sans valeurs	42
Piochez dans votre table avec l'instruction SELECT	43
Révélations sur SQL : confessions d'un NULL	44
Contrôler votre NULL intérieur	45
NOT NULL apparaît dans DESC	47
Remplir les blancs avec DEFAULT	48
Boîte à outils SQL	50

2

l'instruction SELECT

Un talent pour la récupération de données

Est-ce vraiment mieux de donner que de recevoir ? S'il s'agit de bases de données, il y a des chances que vous ayez plus souvent besoin de **récupérer vos données** que de les saisir. C'est là que ce chapitre entre en jeu : vous allez rencontrer la puissante instruction **SELECT** et apprendre à **accéder à ces importantes informations** que vous avez entrées dans vos tables. Vous apprendrez même à utiliser **WHERE**, **AND** et **OR** pour accéder sélectivement à vos données et éviter d'afficher celles dont vous n'avez *pas* besoin.

Rendez-vous ou pas rendez-vous ? 54
Un peu plus SELECT 57
Qu'est-ce que c'est que ce * ? 58
Comment interroger vos types de données 64
Des problèmes de ponctuation supplémentaires 65
Guillemets simples dépareillés 66
Les guillemets simples sont des caractères spéciaux 67
INSERTion de données contenant des apostrophes 68
SELECTionner des colonnes spécifiques pour limiter les résultats 73
SELECTionner des colonnes spécifiques pour des résultats plus rapides 73
Combiner vos requêtes 80
Trouver des valeurs numériques 83
Opérateurs de comparaison ~~sans histoires~~ 86
Trouver des données numériques avec les opérateurs de comparaison 88
Trouver des données texte avec des opérateurs de comparaison 91
Être OU ne pas être 93
La différence entre AND et OR 96
Utiliser IS NULL pour trouver les NULL 99
Gagner du temps avec un seul mot-clé : LIKE 101
Faire appel au joker 101
Seléctionner des rangées en utilisant AND et les opérateurs de comparaison 105
Juste ENTRE nous… on peut faire mieux 106
Après un rendez-vous, soit vous êtes IN... 109
... soit vous êtes NOT IN 110
Encore des NOT 111
Boîte à outils SQL 116

DELETE et UPDATE

3 Le changement vous fera du bien

Vous n'arrêtez pas de changer d'avis ? Maintenant vous pouvez ! Avec les commandes que vous êtes sur le point d'apprendre, **DELETE** et **UPDATE**, vous ne resterez plus coincé avec une décision prise six mois auparavant, quand vous avez commencé à insérer ces données sur les bananes qui, paraît-il, vont bientôt redevenir des coiffures à la mode. Avec UPDATE, vous **pouvez modifier les données** et DELETE vous permet de vous **débarrasser des données** dont vous n'avez plus besoin. Mais nous ne faisons pas que vous donner les outils ; dans ce chapitre, vous apprendrez à être sélectif avec vos nouveaux pouvoirs et à éviter de jeter des données dont vous avez réellement besoin.

Les clowns sont effrayants	120
Sur la piste des clowns	121
Les clowns ont bougé	122
Comment nos données sur les clowns sont-elles saisies ?	126
Bozo, nous avons un problème	128
Se débarrasser d'un enregistrement avec DELETE	129
Utiliser notre nouvelle instruction DELETE	131
Les règles du DELETE	132
Les deux étapes de l'INSERT-DELETE	135
Soyez prudent avec votre DELETE	140
Le problème avec un DELETE imprécis	144
Modifiez vos données avec UPDATE	146
Les règles de l'UPDATE	147
UPDATE est le nouvel INSERT-DELETE	148
UPDATE à l'œuvre	149
Mise à jour des déplacements des clowns	152
UPDATE met à jour vos prix	154
Nous n'avons besoin que d'un UPDATE	156
Boîte à outils SQL	158

conception intelligente de table

4 À quoi bon être normal ?

Vous avez créé des tables sans leur accorder trop de réflexion. Et tout va bien, elles fonctionnent. Vous pouvez utiliser `SELECT`, `INSERT`, `DELETE` et `UPDATE`. Mais **plus vous obtenez de données**, plus vous commencez à voir des **choses que vous auriez aimé avoir fait** pour simplifier vos clauses `WHERE`. Vous avez besoin de rendre vos tables plus *normales*.

Deux tables poissonneuses 160
Une table est centrée sur ses relations 164
Données atomiques 168
Les données atomiques et vos tables 170
Les règles des données atomiques 171
Des raisons d'être normal 174
Les avantages des tables normales 175
Les clowns ne sont pas normaux 176
À mi-chemin de l'1FN 177
Les règles de la clé primaire 178
Vers le NORMAL 181
Réparer la table de Greg 182
Le CREATE TABLE que nous avons écrit 183
Montre-moi l'~~argent~~ la table 184
Une commande pour gagner du temps 185
CREATE TABLE avec une CLÉ PRIMAIRE 186
1, 2, 3... auto incrémentation 188
Ajouter une CLÉ PRIMAIRE à une table existante 192
ALTER TABLE et l'ajout d'une CLÉ PRIMAIRE 193
Boîte à outils SQL 194

ALTER

5 Réécrire le passé

Avez-vous déjà souhaité pouvoir corriger les erreurs de votre passé ? Hé bien, voici votre chance. En utilisant la **commande ALTER**, vous pouvez appliquer toutes les leçons que vous avez apprises à des tables que vous avez conçues il y a des jours, des mois ou même des années. Mieux encore, vous pouvez le faire sans affecter vos données. Quand vous en aurez fini ici, vous saurez ce que la **normalité** signifie vraiment et vous serez capable de l'appliquer à vos tables, passées et présentes.

Nous avons besoin de faire quelques changements	198
Altération de table	203
Rénovation extrême de tables	204
Renommer la table	205
Nous devons faire des plans	207
Retoucher nos colonnes	208
Changement de structure	209
ALTER et CHANGE	210
Changer deux colonnes en une instruction SQL	211
Vite ! Un DROP pour cette colonne	215
La colonne lieu non-atomique vue de plus près	222
Recherche de motifs	223
Quelques fonctions de chaînes pratiques	224
Utiliser une colonne pour en remplir une nouvelle	229
Comment fonctionnent nos combinaisons d'UPDATE et de SET	230
Boîte à outils SQL	232

6

SELECT avancé

Un regard neuf sur vos données

Il est temps d'ajouter un peu de finesse dans votre boîte à outils. Vous savez déjà comment SELECTionner des données et utiliser des clauses WHERE. Mais parfois il faut plus de **précision** que ne peuvent en apporter SELECT et WHERE. Dans ce chapitre, vous apprendrez à **ordonner et regrouper** vos données, ainsi qu'à **lancer des opérations mathématiques** sur vos résultats.

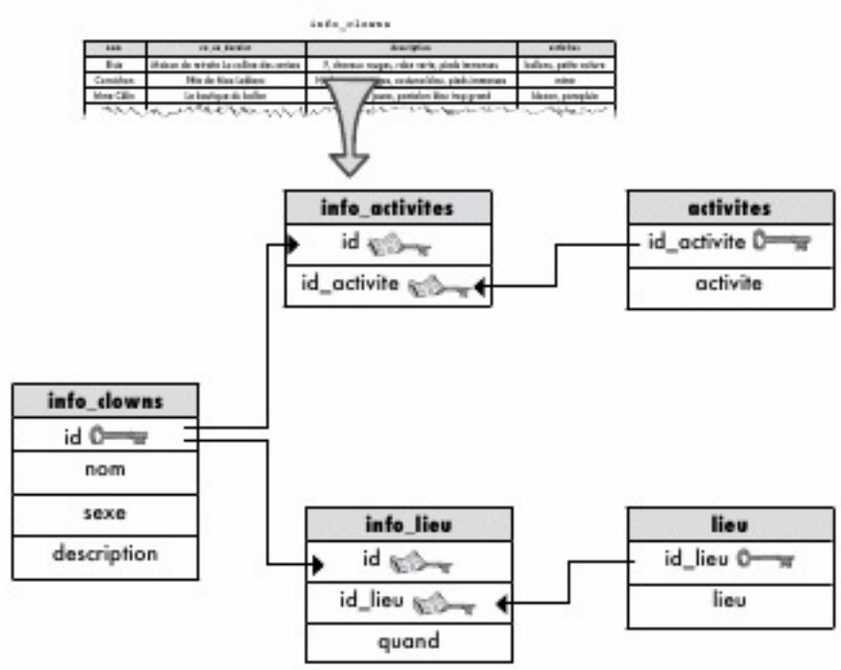
Dataville Vidéo se réorganise 236
Les problèmes de notre table actuelle 237
Faire correspondre les données existantes 238
Peupler la nouvelle colonne 239
UPDATE avec une expression CASE 242
On dirait que nous avons un problème 244
Les tables peuvent se transformer en fouillis 249
Il nous faut un moyen d'organiser les données SELECTionnées 250
Essayez un petit ORDER BY 253
ORDER dans une seule colonne 254
ORDER avec deux colonnes 257
ORDER avec plusieurs colonnes 258
Une table_films ordonnée 259
Inverser l'ORDER avec DESC 261
Le problème de la cheftaine des Jeannettes 263
SUM peut les additionner pour nous 265
SUM tout en un avec GROUP BY 266
AVG avec GROUP BY 267
MIN et MAX 268
COUNT compte les jours 269
SELECT DISTINCT sélectionne des valeurs distinctes 271
LIMITer le nombre de résultats 274
LIMITé à la seconde place 275
Boîte à outils SQL 278

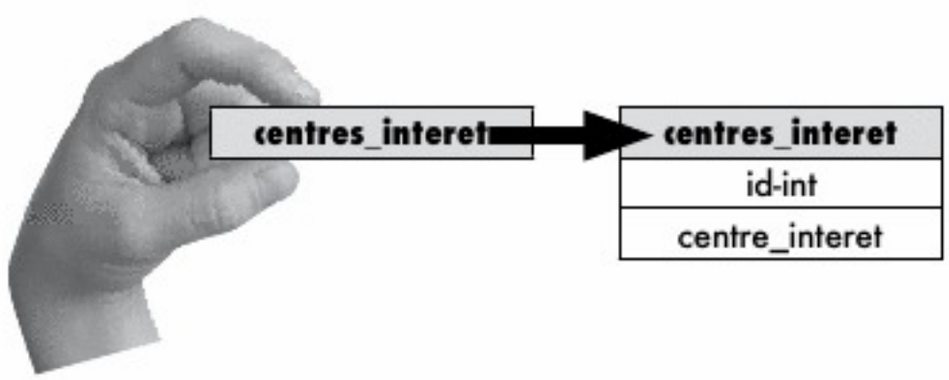
conception de base de données multi-tables

7 Dépasser les limites de votre table

Il arrive qu'une table ne soit plus assez grande. Vos données sont devenues plus complexes et cette **simple table** que vous utilisez ne **fait tout simplement pas le poids**. Votre table est pleine de données redondantes qui gâchent de l'espace et ralentissent vos requêtes. Vous êtes allé aussi loin que vous le pouviez avec une seule table. Le monde est grand et il faut parfois **plus d'une table** pour héberger vos données et les contrôler pour, en fin de compte, être le maître de votre propre base de données.

Trouver une femme pour Nico 282
Tout est perdu… Mais attendez 293
Penser au-delà d'une seule table 294
La base de données multi-tables d'espionnage de clowns 295
Le schéma de la base de données infos_clowns 296
Comment passer d'une table à deux 298
Relier vos tables 303
Contraindre votre clé étrangère 305
L'importance des clés étrangères ? 306
Créer une table avec FOREIGN KEY 307
Relations entre tables 309
Patterns de données : un-à-un 309
Patterns de données : quand utiliser des tables un-à-un 310
Patterns de données : un-à-plusieurs 311
Patterns de données : on arrive à plusieurs-à-plusieurs 312
Patterns de données : nous avons besoin d'une table de jonction 315
Patterns de données : plusieurs-à-plusieurs 316
Enfin en 1FN 321
Les clés composites utilisent plusieurs colonnes 322
Notations abrégées 324
Dépendance fonctionnelle partielle 325
Dépendance fonctionnelle transitive 326
Deuxième forme normale 330
Troisième forme normale (enfin) 336
Et ainsi, Régis (et liste_greg) vécut heureux et eut beaucoup d'enfants 339
Boîte à outils SQL 340





jointures et opérations multi-table

8

Est-ce qu'on ne pourrait pas s'entendre ?

Bienvenue dans un monde multi-table. C'est merveilleux d'avoir **plus d'une table** dans votre base de données, mais vous devez découvrir de ***nouveaux outils et techniques*** pour travailler avec elles. Les tables multiples amènent de la confusion, vous avez donc besoin d'**alias** pour que vos tables restent claires. Et les **jointures** vous aident à relier vos tables, pour que vous puissiez récupérer toutes les données que vous avez éparpillées. Préparez-vous, il est temps de **reprendre le contrôle de votre base de données**.

Nous nous répétons toujours, nous nous répétons... 344
Prépeupler vos tables 345
Nous avons le blues de la table difficile à normaliser 347
La colonne spéciale centres_interet 348
Garder l'intérêt 349
UPDATE met à jour tous vos centres d'intérêt 350
Récupérer tous les centres d'intérêt 351
Plusieurs chemins mènent à Rome 352
CREATE, SELECT et INSERT (presque) en même temps 352
CREATE, SELECT et INSERT en même temps 353
Que change un AS ? 354
Alias de colonnes 355
Des alias de tables, qui aurait besoin de ça ? 356
Tout ce que vous avez toujours voulu savoir sur les jointures internes 357
Jointure cartésienne 358
Libérez votre jointure internet avec INNER JOIN 363
La jointure interne au travail : l'équi-jointure 364
La jointure interne au travail : la non-équi-jointure 367
La dernière jointure interne : la jointure naturelle 368
Des requêtes jointes ? 375
Révélations sur les alias de tables et de colonnes : De quoi vous cachez-vous ? 376
Boîte à outils SQL 377

sous-requetes

9 Des requêtes dans les requêtes

Oui, Jacky, je voudrais une question en deux parties s'il te plaît. Les jointures sont géniales, mais parfois vous avez ***plus d'une question à poser à votre base de données***. Ou vous devez ***prendre le résultat d'une requête pour l'utiliser dans la formulation d'une autre requête***. C'est là que les **sous-requêtes** entrent en jeu. Elles vous aident à **éviter les données dupliquées, elles dynamisent vos requêtes** et elles vous font même entrer dans toutes ces somptueuses fêtes d'après-concert. (En fait, pas vraiment, mais deux sur trois, c'est pas mal !)

Greg débarque sur le marché du recrutement	380
La liste de Greg reçoit des tables supplémentaires	381
Greg utilise une jointure interne	382
Mais il veut essayer d'autres requêtes	384
Sous-requêtes	386
Nous combinons les deux en une requête avec sous-requête	387
Comme si une requête ne suffisait pas : voici la sous-requête	388
Une sous-requête au travail	389
Les règles des sous-requêtes	391
Visite d'une sous-requête en construction	394
Une sous-requête comme colonne de SELECT	397
Un autre exemple : sous-requête avec une jointure naturelle	398
Une sous-requête non-corrélée	399
Révélations sur SQL : Choisir la meilleure requête	400
Une sous-requête non-corrélée avec plusieurs valeurs : IN, NOT IN	403
Sous-requêtes corrélées	408
Une sous-requête corrélée (utile) avec NOT EXISTS	409
EXISTS et NOT EXISTS	410
Le service de recrutement de Greg est ouvert	412
En route pour la fête	413
Boîte à outils SQL	414

requête EXTERNE
requête INTERNE

Requête externe

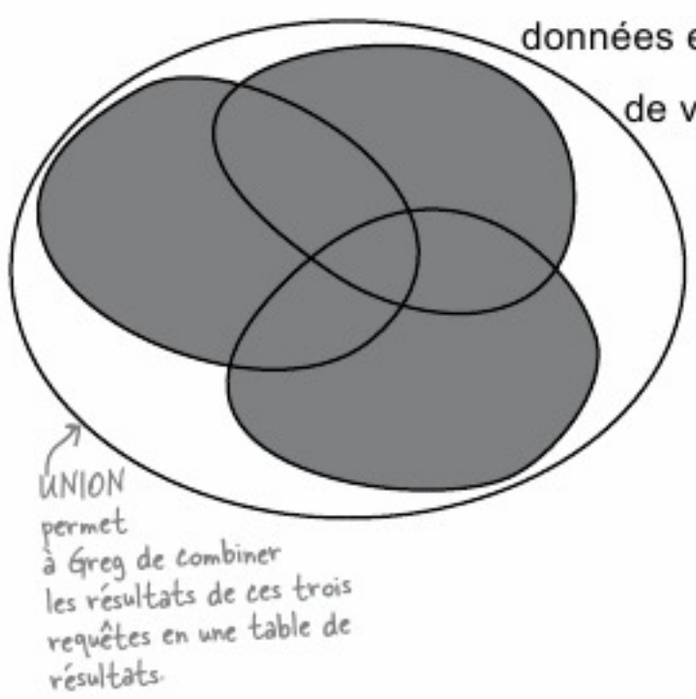
```
SELECT une_colonne, une_autre_colonne
FROM table
WHERE colonne = (SELECT colonne FROM table);
```

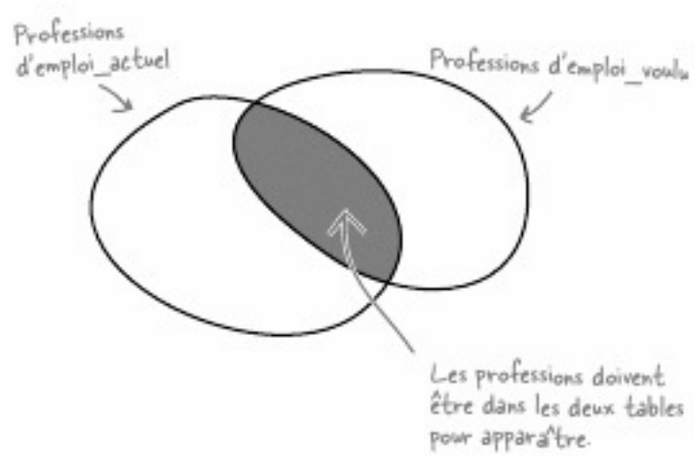
Requête interne

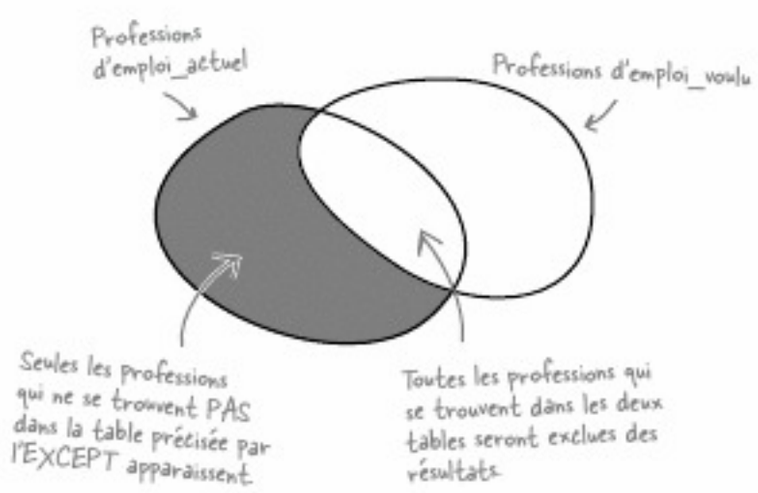
jointures externes, auto-jointures et unions

10 Nouvelles tactiques

Vous ne connaissez que la moitié de ce qu'il y a à savoir sur les jointures. Vous avez vu les jointures croisées qui renvoient toutes les lignes possibles et les jointures internes qui renvoient les lignes des deux tables là où il y a correspondance. Mais vous n'avez pas vu les **jointures externes** qui vous renvoient des lignes qui *n'ont pas de correspondances dans l'autre table*, **les auto-jointures** qui (assez bizzarement) *joignent une table à elle-même* et les **unions** qui *combinent les résultats des requêtes*. Une fois que vous aurez appris ces astuces, vous pourrez récupérer vos données exactement de la façon qu'il vous faut. (Et nous n'avons pas non plus oublié de vous révéler la vérité sur les sous-requêtes !)







Nettoyage des vieilles données	418
À propos de gauche et droite	419
Voici une jointure externe gauche	420
Jointures externes et correspondances multiples	425
La jointure externe droite	426
Pendant que vous faisiez des jointures externes…	429
On peut créer une nouvelle table	430
Comment s'intégre la nouvelle table	431
Une clé étrangère auto-référencée	432
Joindre une table à elle-même	433
Nous avons besoin d'une auto-jointure	435
Un autre moyen d'obtenir des informations multi-tables	436
Vous pouvez utiliser une UNION	437
UNION est limitée	438
Les règles de l'UNION en action	439
UNION ALL	440
Créer une table à partir d'une union	441
INTERSECT et EXCEPT	442
Nous en avons fini avec les jointures, il est temsp de passer à…	443
Comparaisons entre jointures et sous-requêtes	443
Changer une sous-requête en jointure	444
Une auto-jointure comme sous-requête	449
L'entreprise de Greg se développe	450
Boîte à outils SQL	452

contraintes, vues et transactions

11 Trop de cuisiniers gâtent la base de données

Votre base de données a grandi et d'autres que vous doivent l'utiliser. Le problème est que certains d'entre eux ne sont pas aussi compétents que vous en SQL. Vous avez besoin de moyens de **les empêcher de saisir de mauvaises données**, de techniques pour ne les autoriser à **voir qu'une partie des données** et de méthodes pour **les empêcher de se marcher les uns sur les autres quand ils essaient d'entrer des données en même temps**. Dans ce chapitre, nous commençons à protéger vos données des erreurs des autres. Bienvenue dans les Bases de données défensives, 1ère partie.

Greg a recruté de l'aide	456
Le premier jour de Martin : insertion d'un nouveau client	457
Martin évite un NULL	458
Bond de trois mois dans le futur	459
Par CHECK, s'il vous plaît : ajouter une CONTRAINTE CHECK	460
CHECK pour vérifier le sexe	461
Le travail de Norbert devient fastidieux	463
Créer une vue	465
Voir vos vues	466
Ce que fait vraiment votre vue	467
Ce qu'est une vue	468
INSERT, UPDATE et DELETE avec des vues	471
Le secret est de prétendre qu'une vue est une vraie table	472
Vue avec CHECK OPTION	475
Votre vue est modifiable si...	476
Quand vous en avez terminé avec votre vue	477
Quand il arrive du mal à de bonnes bases de données	478
Ce qui s'est passé dans le distributeur	479
D'autres problèmes au distributeur	480
Ce n'est pas un rêve, c'est une transaction	482
Le test classique de l'ACID	483
SQL vous aide à gérer vos transactions	484
Ce qui aurait dû se passer dans le distributeur	485
Comment faire fonctionner des transactions avec MySQL	486
Maintenant essayez-le vous-même	487
Boîte à outils SQL	490

sécurité

12 Protéger vos biens

Vous avez investi une quantité énorme de temps et d'énergie pour créer votre base de données. Et vous seriez désolé si quelque chose lui arrivait. Vous avez aussi dû donner **accès à vos données** à d'autres personnes et vous avez peur qu'ils utilisent mal un insert ou un update, ou pire encore, un **delete**. Vous êtes sur le point d'apprendre comment les bases de données et les objets qui s'y trouvent peuvent devenir plus **sécurisés** et comment vous pouvez avoir un contrôle total sur ***qui fait quoi avec vos données.***

Problèmes d'utilisateur 494
Eviter les erreurs dans la base de données d'espionnage des clowns 495
Protéger le compte utilisateur root 497
Ajouter un nouvel utilisateur 498
Décider exactement de quoi l'utilisateur a besoin 499
Une simple instruction GRANT 500
Variations sur GRANT 503
REVOKE révoque les privilèges 504
RÉVOQUER une GRANT OPTION déjà utilisée 505
REVOKE avec précision 506
Le problème des comptes partagés 510
Utiliser votre rôle 512
Suppression de rôle 512
Utiliser votre rôle WITH ADMIN OPTION 514
Combiner CREATE USER et GRANT 519
La liste de Greg devient internationale ! 520
Boîte à outils SQL 522
Que diriez-vous d'une Liste de Greg dans votre ville ? 524
Utilisez SQL dans vos projets et vous aussi vous pourriez être comme Greg ! 524

root

timide

doc

simplet

grincheux

joyeux

dormeur

atchoum

A les restes

Les dix sujets les plus importants (dont nous n'avons pas parlé)

Même après tout cela, il en reste encore un peu. Il y a juste quelques choses en plus que nous pensons que vous devez savoir. Cela nous gênerait de les ignorer, même si elles ne méritent qu'une simple évocation. Alors avant de reposer le livre, parcourez ces **courts mais importants petits bijoux du SQL**.

Par ailleurs, une fois que vous aurez fini, il ne restera que deux annexes... et l'index... et peut-être un peu de pub... et alors vous aurez vraiment terminé. Promis !

n°1. Une interface graphique pour votre SGBDR 526
n°2. Mots réservés et caractères spéciaux 528
n°3. ALL, ANY et SOME 530
n°4. Plus d'infos sur les types de données 532
n°5. Tables temporaires 534
n°6. Lancez CAST sur vos données 535
n°7. Qui êtes-vous ? Quelle heure est-il ? 536
n°8. Fonctions numériques utiles 537
n°9. Indexer pour accélérer les choses 539
n°10. 2 minutes de PHP/MySQL 540

A	ABSOLUTE ACTION ADD ADMIN AFTER AGGREGATE ALIAS ALL ALLOCATE ALTER AND ANY ARE ARRAY AS ASC ASSERTION AT AUTHORIZATION
B	BEFORE BEGIN BINARY BIT BLOB BOOLEAN BOTH BREADTH BY
C	CALL CASCADE CASCADED CASE CAST CATALOG CHAR CHARACTER CHECK CLASS CLOB CLOSE COLLATE COLLATION COLUMN COMMIT COMPLETION CONNECT CONNECTION CONSTRAINT CONSTRAINTS CONSTRUCTOR CONTINUE CORRESPONDING CREATE CROSS CUBE CURRENT CURRENT_DATE CURRENT_PATH CURRENT_ROLE CURRENT_TIME CURRENT_TIMESTAMP CURRENT_USER CURSOR CYCLE
D	DATA DATE DAY DEALLOCATE DEC DECIMAL DECLARE DEFAULT DEFERRABLE DEFERRED DELETE DEPTH DEREF DESC DESCRIBE DESCRIPTOR DESTROY DESTRUCTOR DETERMINISTIC DICTIONARY DIAGNOSTICS DISCONNECT DISTINCT DOMAIN DOUBLE DROP DYNAMIC
E	EACH ELSE END END_EXEC EQUALS ESCAPE EVERY EXCEPT EXCEPTION EXEC EXECUTE EXTERNAL
F	FALSE FETCH FIRST FLOAT FOR FOREIGN FOUND FROM FREE FULL FUNCTION
G	GENERAL GET GLOBAL GO GOTO GRANT GROUP GROUPING
H	HAVING HOST HOUR
I	IDENTITY IGNORE IMMEDIATE IN INDICATOR INITIALIZE INITIALLY INNER INOUT INPUT INSERT INT INTEGER INTERSECT INTERVAL INTO IS ISOLATION ITERATE
J	JOIN
K	KEY
L	LANGUAGE LARGE LAST LATERAL LEADING LEFT LESS LEVEL LIKE LIMIT LOCAL LOCALTIME LOCALTIMESTAMP LOCATOR
M	MAP MATCH MINUTE MODIFIES MODIFY MODULE MONTH
N	NAMES NATIONAL NATURAL NCHAR NCLOB NEW NEXT NO NONE NOT NULL NUMERIC
O	OBJECT OF OFF OLD ON ONLY OPEN OPERATION OPTION OR ORDER ORDINALITY OUT OUTER OUTPUT
P	PAD PARAMETER PARAMETERS PARTIAL PATH POSTFIX PRECISION PREFIX PREORDER PREPARE PRESERVE PRIMARY PRIOR PRIVILEGES PROCEDURE PUBLIC
Q	
R	READ READS REAL RECURSIVE REF REFERENCES REFERENCING RELATIVE RESTRICT RESULT RETURN RETURNS REVOKE RIGHT ROLE ROLLBACK ROLLUP ROUTINE ROW ROWS
S	SAVEPOINT SCHEMA SCROLL SCOPE SEARCH SECOND SECTION SELECT SEQUENCE SESSION SESSION_USER SET SETS SIZE SMALLINT SOME SPACE SPECIFIC SPECIFICTYPE SQL SQLEXCEPTION SQLSTATE SQLWARNING START STATE STATEMENT STATIC STRUCTURE SYSTEM_USER
T	TABLE TEMPORARY TERMINATE THAN THEN TIME TIMESTAMP TIMEZONE_HOUR TIMEZONE_MINUTE TO TRAILING TRANSACTION TRANSLATION TREAT TRIGGER TRUE
U	UNDER UNION UNIQUE UNKNOWN UNNEST UPDATE USAGE USER USING
V	VALUE VALUES VARCHAR VARIABLE VARYING VIEW
W	WHEN WHENEVER WHERE WITH WITHOUT WORK WRITE
X	
Y	YEAR
Z	ZONE

```
Fichier Edition Fenêtre Aide
> SELECT CURRENT_DATE;
+--------------+
| CURRENT_DATE |
+--------------+
| 2007-07-26   |
+--------------+
1 row in set (0.00 sec)
```

```
Fichier Edition Fenêtre Aide
> SELECT CURRENT_TIME;
+--------------+
| CURRENT_TIME |
+--------------+
| 11:26:48     |
+--------------+
1 row in set (0.00 sec)
```

```
Fichier Edition Fenêtre Aide
SELECT CURRENT_USER;
+----------------+
| CURRENT_USER   |
+----------------+
| root@localhost |
+----------------+
1 row in set (0.00 sec)
```

B

installation de mySQL

Voyez par vous-même

Tous vos nouveaux talents en SQL ne vous apporteront pas grand chose sans un endroit où les exercer. Cette annexe contient les instructions pour installer votre propre SGBDR MySQL et pouvoir travailler dessus.

Pouvoir commencer, vite ! 544

Instructions et dépannage 544

Étapes pour installer MySQL sous Windows 545

Étapes pour installer MySQL sous Mac OS X 548

C

au rendez-vous des outils

Tous vos nouveaux outils SQL

Voici tous vos outils SQL réunis au même endroit pour la première fois, pour une unique soirée ! Plus sérieusement, vous êtes au point de rendez-vous de tous les outils SQL que nous avons traités. Prenez un moment pour *examiner la liste et vous sentir* ***fier*** : **vous les avez tous appris** !

Symboles 552

A–B 552

C 553

D-E 554

F–L 555

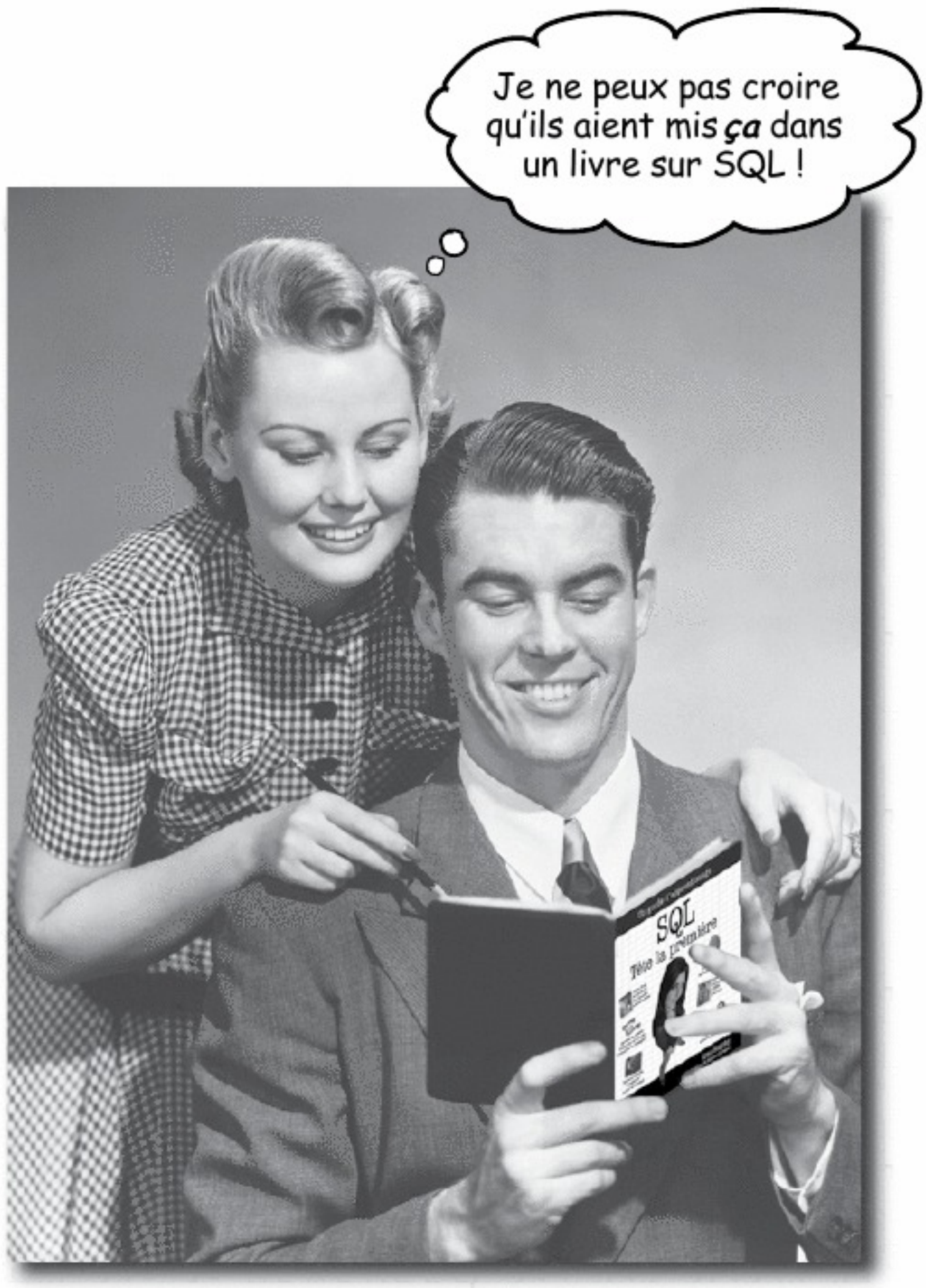
L–R 556

R–S 557

S–V 558

V–W 559

Intro



C'est dans ce chapitre que nous allons répondre à la question qui vous brûle les lèvres : « mais comment ont-ils osé mettre ça dans un livre sur SQL ? »

À qui s'adresse ce livre ?

Si vous répondez « oui » à toutes ces questions :

1. Avez-vous accès à un ordinateur avec un SGBDR, comme Oracle, MS SQL ou MySQL ? Ou sur lequel vous pouvez installer MySQL ou d'autres SGBDR ?
2. Voulez-vous **apprendre**, **comprendre et retenir comment créer des tables, des bases de données et écrire des requêtes en utilisant les standards les meilleurs et les plus récents ?**
3. Préférez-vous les **discussions vivantes** aux **leçons scolaires ennuyeuses** ?

Ce livre est fait pour vous.

Nous allons vous aider à apprendre les concepts et la syntaxe SQL d'une façon qui vous facilite la compréhension et la pratique de SQL pour pouvoir l'utiliser exactement comme vous le voulez.

À qui n'est-il pas destiné ?

Si vous répondez « oui » à l'une de ces questions :

1. Êtes-vous **absolument à l'aise** avec la syntaxe SQL de base et à la recherche de quelque chose qui vous aide pour la conception avancée de bases de données ?
2. Êtes-vous un programmeur SQL confirmé à la recherche d'un ***livre de référence*** sur SQL ?
3. Avez-vous **peur d'essayer quelque chose de différent** ? Vous préféreriez vous faire dévitaliser une dent plutôt que de mélanger les carreaux et les rayures ? Croyez-vous qu'un livre technique ne peut pas être sérieux si les concepts de programmation sont présentés sous forme humaine ?

Mais si vous voulez vous rafraîchir la mémoire et n'avez jamais vraiment compris les formes normales, les relations un-à-plusieurs et les jointures externes gauches, ce livre peut vous aider.

Ce livre n'est pas pour vous.

[Note du marketing : ce livre est destiné à quiconque possède une carte bleue.]

Nous savons ce que vous pensez

« Ce livre est tout sauf sérieux »

« C'est quoi toutes ces images ? »

« On peut apprendre de cette façon ? »

Et nous savons ce que pense votre *cerveau*

Votre cerveau est avide de nouveauté. Toujours à l'affût, il ne cesse de chercher et d'attendre quelque chose d'inhabituel. C'est comme ça qu'il est construit, et c'est comme ça qu'il vous aide à rester en vie.

Aujourd'hui, il est peu probable que vous serviez de petit-déjeuner à un tigre. Mais votre cerveau reste sur ses gardes. On ne sait jamais…

Alors que fait-il de toutes les choses ordinaires, routinières et banales que vous rencontrez ? Il fait tout ce qu'il peut pour les empêcher d'interférer avec son vrai travail : enregistrer ce qui compte. Il ne se soucie pas de mémoriser les choses ennuyeuses ; il considère a priori qu'elles n'ont « aucune importance » et il les filtre.

Mais comment votre cerveau sait-il ce qui est important ? Supposons que vous partiez en excursion pour la journée et qu'un tigre surgisse devant vous. Que se passe-t-il dans votre tête ?

Les neurones s'embrasent. Les émotions vous submergent. Les hormones déferlent.

Et votre cerveau ne sait rien d'autre…

Cela peut être important ! Ne l'oubliez pas !

Mais imaginez que vous êtes chez vous ou dans une bibliothèque. C'est un lieu sûr et chaleureux. Pas le moindre tigre à l'horizon… Vous êtes en train d'étudier. De réviser un examen. Ou d'essayer de comprendre un sujet bien technique dont votre patron vous a dit qu'il vous prendra huit à dix jours au plus.

Seul problème : votre cerveau essaie de vous faire un gros cadeau. Il essaie de faire en sorte que des contenus évidemment sans importance n'épuisent pas des ressources déjà rares. Des ressources mieux employées à mémoriser les choses vraiment essentielles. Comme les tigres. Comme les incendies. Comme le danger qu'il y a à faire du ski en short.

Et il n'y a a aucun moyen simple de lui dire : « Hé, cerveau, merci bien, mais même si ce livre est ennuyeux comme la pluie, et même si j'ai le moral à zéro, je veux que tu fasses attention. »

Nous considérons notre lecteur comme un apprenant

Comment *apprend-on* quelque chose ? Il faut d'abord *comprendre*, puis faire en sorte de ne pas *oublier*. Il ne s'agit pas de remplir sa tête de force. Les dernières recherches en sciences cognitives, en neurobiologie et en psychologie de l'éducation montrent que l'*apprentissage* exige autre chose que du texte imprimé. Nous savons ce qui stimule votre cerveau.

Voici quelques-uns de nos principes :

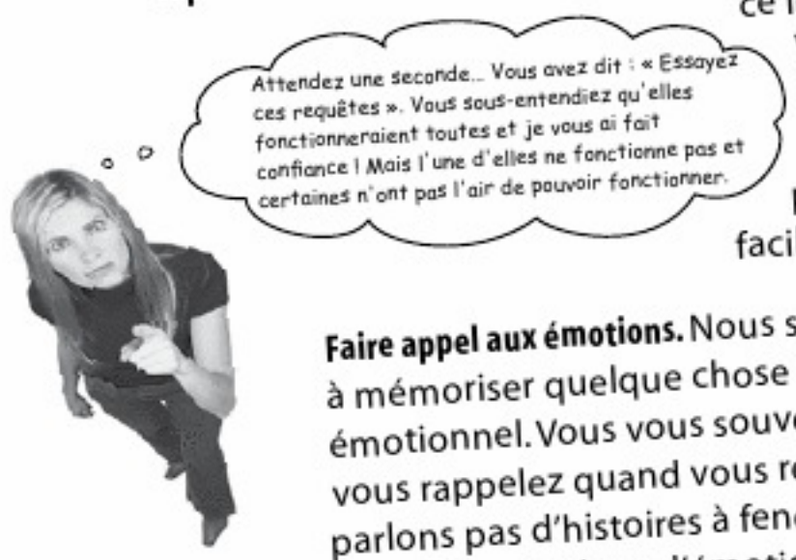
Permettre de visualiser. Les images sont beaucoup plus faciles à mémoriser que les mots et facilitent beaucoup l'apprentissage (jusqu à 89 % d'amélioration dans les études sur le rappel et le transfert). Elles aident également à mieux comprendre.

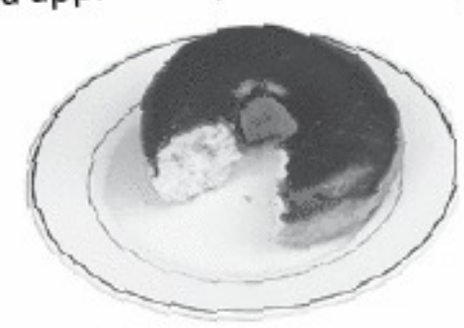
Placer le texte près de l'image auquel il se rapporte, et non à la fin de la page ou plus loin, permet aux apprenants de résoudre deux fois plus vite les problèmes liés aux contenus.

Adopter un style conversationnel et personnalisé. De récentes expériences ont montré que des étudiants se rappelaient beaucoup mieux un texte s'adressant directement à eux sur le ton de la conversation que des sujets qui avaient lu un texte présenté de façon plus formaliste. Leurs performances étaient supérieures de 40 %. Mieux vaut raconter une histoire que pontifier, parler le langage de tous les jours, éviter de se prendre trop au sérieux. À qui prêtez-vous le plus d'attention : à l'ami avec qui vous dînez ou à un conférencier ?

Faire réfléchir. Si vous n'exercez pas activement vos neurones, il ne se passe pas grand chose dans votre tête. Un lecteur doit être motivé, engagé, curieux. Il doit avoir envie de résoudre des problèmes, de tirer des conclusions et de générer de nouvelles connaissances. Pour ce faire, il a besoin de défis, d'exercices, de questions qui incitent à réfléchir et d'activités qui sollicitent les deux hémisphères cérébraux et font appel à plusieurs sens.

Capter — et conserver — l'attention du lecteur. Nous avons tous pensé un jour « Je veux vraiment lire ce livre, mais il me tombe des mains dès la première page ». Votre cerveau prête attention à ce qui sort de l'ordinaire, qui attire l'œil, à ce qui est intéressant, étrange, inattendu. L'étude d'un nouveau sujet technique, même difficile, n'a pas besoin d'être fastidieuse. Votre cerveau apprendra plus facilement si vous ne vous ennuyez pas.





Faire appel aux émotions. Nous savons maintenant que notre aptitude à mémoriser quelque chose dépend largement de son contenu émotionnel. Vous vous souvenez de ce qui compte pour vous. Vous vous rappelez quand vous ressentez quelque chose. Non, nous ne parlons pas d'histoires à fendre le cœur avec un petit garçon et son chien. Nous parlons d'émotions comme la surprise, la curiosité, l'amusement ou la sensation de toute-puissance que vous éprouvez lorsque vous résolvez un puzzle, que vous apprenez quelque chose que tout le monde considère comme difficile, ou lorsque vous vous rendez compte que vous savez quelque chose que les super-gourous ne savent pas.

La métacognition : apprendre à apprendre

Si vous voulez réellement apprendre, plus vite et plus en profondeur, soyez attentif à la manière dont vous faites attention. Réfléchissez à la façon dont vous réfléchissez. Apprenez à apprendre.

La plupart d'entre nous n'ont pas suivi de cours sur la métacognition ou la théorie de l'apprentissage lorsque nous étions à l'école. Nous étions *censés* apprendre, mais on nous disait rarement comment faire.

Mais si vous avez ce livre entre les mains, nous supposons que vous voulez apprendre SQL. Et vous ne voulez probablement pas y consacrer une éternité. Et vous voudrez aussi vous souvenir de se que vous avez lu et pouvoir ensuite l'appliquer. Et pour cela il faut *comprendre*. Pour tirer le meilleur parti de cet ouvrage, d'un autre livre ou de toute expérience d'apprentissage, apprenez à maîtriser votre cerveau.

L'astuce consiste à l'amener à considérer ce que vous allez apprendre comme quelque chose de réellement important, de capital pour votre bien-être, d'aussi important qu'un tigre. Sinon, vous êtes voué à un combat sans fin : votre cerveau fera tout son possible pour vous empêcher de mémoriser.

Mais comment *FAIRE* au juste pour que votre cerveau traite SQL comme un tigre affamé ?

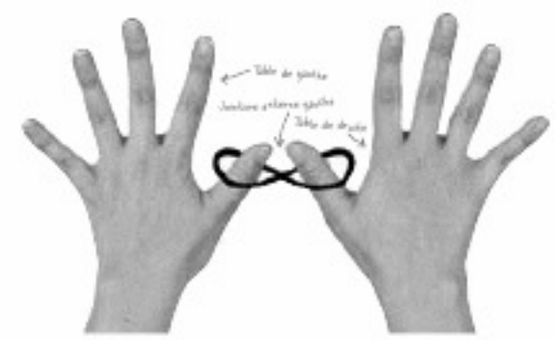
Il y a deux façons de procéder : l'une lente et ennuyeuse, l'autre rapide et efficace. La solution lente consiste purement et simplement à répéter. Vous savez certainement que vous êtes capable d'apprendre et de mémoriser le plus ingrat des sujets si vous répétez sans arrêt la même chose. Au bout d'un nombre suffisant de répétitions, votre cerveau pense : « Ça ne me semble pas important pour lui, mais s'il ressasse ce truc, c'est que ça doit l'être. »

La voie rapide consiste à faire ***tout ce qui non seulement augmente mais aussi diversifie l'activité cérébrale***. Les éléments de la page précédente constituent une grande partie de la solution, et ils se sont tous révélés capables d'aider votre cerveau à travailler en votre faveur. Certaines études démontrent que l'insertion de mots dans les images qu'ils décrivent (et non dans des légendes ou dans le corps du texte) force le cerveau à tenter de comprendre les relations entre le texte et l'image, et provoque des connexions neuronales plus nombreuses. Plus ces connexions se multiplient, plus il a de chances de comprendre que le contenu est digne d'attention et mérite d'être retenu.

Un style conversationnel aide également. Les gens on tendance à faire plus attention à une conversation, puisqu'ils sont censés la suivre et éventuellement y prendre part. Chose surprenante : votre cerveau ne se soucie pas nécessairement de savoir que vous « conversez » avec un livre ! En revanche, si le style est aride et formaliste, il le perçoit exactement comme celui d'un conférencier pérorant devant une salle pleine d'auditeurs passifs. Inutile en ce cas de rester éveillé.

Mais les graphismes et le ton de la conversation ne sont qu'un début.

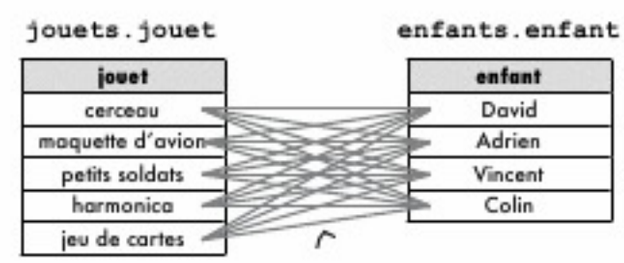
Ce que NOUS avons fait :



Nous avons utilisé des ***images***, parce que votre cerveau les préfère au texte. Pour lui, une seule d'entre elles vaut réellement mieux que 1024 mots. Et, lorsque texte et image vont de pair, nous avons fusionné le texte et l'image, parce que votre cerveau fonctionne mieux lorsque les mots sont intégrés à ce qu'il décrivent plutôt que placés dans une légende ou enfouis quelque part dans la page imprimée.

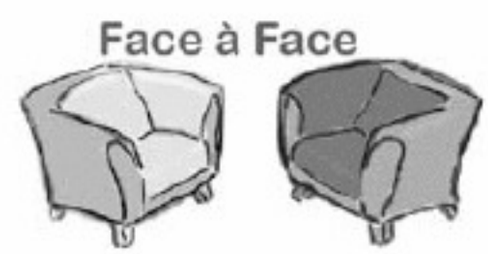
Nous avons utilisé la ***répétition***, en exprimant la même chose de différentes manières, en recourant à divers types de supports et en faisant appel à plusieurs sens, pour augmenter les chances que le contenu soit encodé dans plusieurs zones de votre cerveau.

Nous avons utilisé concepts et images de façon ***inattendue***, parce que votre cerveau aime la nouveauté. Nous avons ajouté au moins un peu de contenu ***émotionnel***, parce que votre cerveau prête attention à la biochimie des émotions. C'est ce qui vous aide à mémoriser, même si ce n'est dû à rien d'autre qu'un peu d'***humour***, de ***surprise*** ou d'***intérêt***.



Nous avons utilisé un ***style conversationnel***, personnel, parce que votre cerveau est plus enclin à être attentif quand il croit que vous êtes engagé dans une conversation que lorsqu'il pense que vous assistez passivement à une présentation. Il fait de même quand vous lisez.

Nous avons inclus plus de 80 ***exercices***, parce que votre cerveau apprend et retient mieux quand vous faites quelque chose que lorsque vous lisez. Ces exercices sont « difficiles mais faisables » parce que c'est ce que la plupart des gens préfèrent.



Nous avons fait appel à ***plusieurs styles d'apprentissage***, parce que vous préférez peut-être les procédures pas-à-pas, alors que quelqu'un d'autre voudra d'abord saisir l'image globale et qu'un troisième lecteur cherchera simplement un exemple de code. Mais, indépendamment de ses préférences, chacun peut tirer parti de la vision d'un même contenu diversement présenté.

Nous avons inclus des contenus qui s'adressent aux ***deux hémisphères de votre cerveau*** : plus ce dernier est engagé, plus vous êtes à même d'apprendre, de mémoriser et de vous concentrer plus longtemps. Comme faire travailler l'un des côtés du cerveau permet souvent à l'autre de se reposer, la période durant laquelle votre apprentissage est productif est plus longue.

Nous avons également incorporé des ***histoires*** et des ***exercices*** qui présentent plusieurs ***points de vue***, parce que votre cerveau est ainsi fait que vous apprenez plus en profondeur s'il est forcé de faire des évaluations et de porter des jugements.

Enfin, nous avons inclus des ***défis*** dans les exercices en posant des ***questions*** qui n'ont pas de réponse simple, parce que votre cerveau apprend et retient mieux lorsqu'il doit travailler (tout comme il ne suffit pas de regarder les autres à la gym pour rester en bonne forme physique). Mais nous avons fait de notre mieux pour que, si vous devez travailler dur, ce soit sur les bons éléments : pour que vous ne consacriez pas un neurone de trop à traiter un exemple difficile à comprendre, ni à analyser un texte difficile, bourré de jargon et horriblement laconique.

Nous avons fait appel à des ***personnes*** dans les histoires, dans les exemples et dans les images, parce que vous êtes une personne. Et votre cerveau s'intéresse plus aux personnes qu'aux choses.

Voici ce que VOUS pouvez faire pour dompter votre cerveau

Nous avons donc fait notre part. Le reste dépend de vous. Les conseils qui suivent sont un point de départ. Écoutez votre cerveau et déterminez ce qui fonctionne pour vous et ce qui ne fonctionne pas. Essayez de nouvelles techniques.

À découper et à coller sur le frigo

1. **Prenez votre temps. Plus vous comprenez, moins vous avez à mémoriser.**
   Ne vous bornez pas à lire. Arrêtez-vous et réfléchissez. Quand vous voyez une question, ne vous précipitez pas sur la réponse. Imaginez que quelqu'un la pose réellement. Plus vous forcez votre cerveau à penser, plus vous avez de chances d'apprendre et de retenir.

2. **Faites les exercices. Prenez des notes.**
   Si nous les avions faits pour vous, ce serait comme de demander à quelqu'un de faire de la gym à votre place. Et ne vous contentez pas de les lire. **Prenez un crayon.** Tout montre que l'activité physique *durant* l'apprentissage permet d'apprendre plus.

3. **Lisez les « Questions Stupides ».**
   Et lisez tout. Ce ne sont pas des encadrés facultatifs : ***ils font partie du contenu !*** Parfois, les questions sont plus utiles que les réponses.

4. **Faites de ce livre votre dernière lecture avant de dormir. Au moins la dernière chose *difficile*.**
   Une partie de l'apprentissage (surtout le transfert dans la mémoire à long terme) aura lieu *après* que vous aurez reposé le livre. Votre cerveau a besoin de temps à lui pour travailler plus. Si vous interposez quelque chose de nouveau dans ce laps de temps, une partie de ce que vous viendrez d'apprendre sera perdue.

5. **Buvez beaucoup d'eau.**
   Votre cerveau fonctionne mieux lorsqu'il est convenablement irrigué. La déshydratation (qui peut apparaître avant la soif) diminue les fonctions cognitives.

6. **Parlez-en à voix haute.**
   La parole active une autre partie du cerveau. Si vous essayez de comprendre quelque chose ou d'augmenter vos chances de vous le remémorer plus tard, dites-le à haute voix. Mieux encore, essayez de l'expliquer à quelqu'un. Vous apprendrez plus vite et vous découvrirez peut-être des idées qui vous ont échappé à la lecture.

7. **Écoutez votre cerveau.**
   Veillez à ne pas le surcharger. Si vous vous surprenez à lire en diagonale ou à oublier ce que vous venez de lire, il est temps de faire une pause. Passé un certain point, vous n'apprenez pas plus vite en essayant d'en absorber davantage, et vous risquez même d'altérer le processus.

8. **Ressentez quelque chose !**
   Votre cerveau a besoin de savoir que c'est important. Impliquez-vous dans les histoires. Créez vos propres légendes pour les photos. S'agacer d'une mauvaise plaisanterie vaut mieux que ne rien ressentir du tout.

9. ***Concevez* quelque chose !**
   Appliquez ce que vous lisez à quelque chose que vous concevez ou retravaillez un vieux projet. Faites simplement quelque chose pour vous construire de l'expérience au-delà des exercices et des activités de ce livre. Tout ce qu'il vous faut est un problème à résoudre... un problème qui pourrait bénéficier d'une ou de plusieurs des techniques dont nous allons parler.

Lisez-moi

Il s'agit d'un livre d'initiation, non d'un ouvrage de référence. Nous avons délibérément éliminé tout ce qui pouvait faire obstacle à l'apprentissage, quel que soit le sujet abordé à un moment donné. Et, lors de votre première lecture, vous devrez commencer par le commencement, parce que ce livre tient compte de ce que vous avez déjà vu et appris.

Nous commençons par enseigner la syntaxe SQL de base, puis des concepts de conception de bases de données et enfin les requêtes avancées.

Même s'il est important de créer des bases de données et des tables bien conçues, avant de pouvoir le faire, vous devez comprendre la syntaxe SQL. Nous commençons donc par vous donner des instructions SQL que vous pouvez essayer concrètement. Ainsi vous pouvez tout de suite faire quelque chose avec SQL et vous commencerez à vous y intéresser. Ensuite, un peu plus tard dans le livre, nous vous montrons de bonnes habitudes de conception de tables. À ce moment-là vous aurez une connaissance solide de la syntaxe dont vous avez besoin et vous pourrez vous concentrer sur *l'apprentissage des concepts*.

Nous ne traitons pas de tous les mots-clés, fonctions et instructions SQL.

Nous aurions pu mettre tous les mots-clés, fonctions et instructions SQL dans ce livre, mais nous avons pensé que vous préfèreriez avoir un livre relativement transportable qui vous enseigne les mots-clés, fonctions et instructions les plus importants. Nous vous présentons ceux que vous avez besoin de connaître, ceux que vous utiliserez 95% du temps. Et quand vous aurez fini ce livre, vosu aurez l'assurance d'aller chercher la fonction dont vous avez besoin pour terminer cette requête du tonnerre que vous venez d'écrire.

Nous ne traitons pas tous les SGBDR.

Il existe Standard SQL, MySQL, Oracle, MS SQL Server, PostgreSQL, DB2 et encore quelques autres SGBDR. Si nous traitions toutes les variations de syntaxe pour toutes les commandes du livre, ce livre aurait beaucoup plus de pages. Comme nous aimons les arbres, nous nous concentrons sur Standard SQL avec a crochet vers MySQL. Tous les exemples du livre fonctionnent avec MySQL. Et la plupart fonctionneront avec les SGBDR listés ci-dessus. Vous vous souvenez de ce livre de référence que nous vous avons suggéré d'acheter ? Achetez-en un pour le SGBDR que vous utilisez.

Les activités ne sont PAS facultatives.

Les exercices et les activités ne sont pas là pour décorer : ils sont une partie intrinsèque de ce livre. Certains sont là pour vous aider à mémoriser, d'autres à comprendre et d'autres encore pour vous aider à appliquer ce que vous avez appris. ***Ne sautez pas les exercices***. Les mots-croisés ne sont pas obligatoires mais ils sont conseillés afin que vous puissiez penser à ces mots dans un autre contexte.

La redondance est intentionnelle et importante.

Un livre Tête la première possède un trait distinctif : nous voulons réellement que vous compreniez et que vous vous souveniez de ce que vous avez appris lorsque vous le refermez. La plupart des ouvrages de référence ne sont pas écrits dans cet objectif, mais, dans celui-ci, il s'agit d'apprendre. C'est pourquoi vous verrez les mêmes concepts abordés plusieurs fois.

Les exemples de code sont aussi courts que possible.

Nos lecteurs nous disent à quel point il est frustrant de devoir parcourir 200 lignes de code pour trouver les deux instructions qu'ils ont besoin de comprendre. Ici, la plupart des exemples sont présentés dans le contexte le plus réduit possible, pour que la partie que vous voulez apprendre demeure simple et claire. Ne vous attendez donc pas à ce que le code soit robuste, ni même complet. Les exemples sont spécifiquement conçus pour l'apprentissage et ne sont pas toujours pleinement fonctionnels.

Nous avons placé de nombreuses commandes sur le web pour que vous puissiez les copier-coller dans votre terminal ou dans le logiciel de votre base de données. Vous les trouverez en français sur :
http://tetelapremiere.fr/livres/sql

Les exercices « Musclez votre cerveau » n'ont pas de réponse.

Pour certains, il n'existe pas de « bonne réponse ». Pour d'autres, c'est à vous qu'il appartient de décider si vos réponses sont justes ou non. Dans certains de ces exercices, vous trouverez des indications destinées à vous mettre sur la bonne voie.

L'équipe technique

Nos épatants relecteurs :

Nous sommes extrêment redevable à notre équipe de relecteurs techniques. Ils ont repéré de nombreuses erreurs, flagrantes ou subtiles, et des typos pathétiques. Sans eux, ce livre serait loin d'être aussi propre et correct. Ils ont réalisé le minutieux travail d'élimination des erreurs de ce livre.

Cary Collett a mis ses 15 ans d'expérience de travail pour des start-ups, des ateliers du gouvernement et actuellement pour le secteur financier au service de la relecture de ce livre et est impatient de retourner profiter de ses activités extra-professionnelles comme cuisiner, faire des randonnées, lire et terroriser ses chiens.

LuAnn Mazza a trouvé le temps, dans sa vie professionnelle très occupée d'analyste et développeur de logiciel dans l'Illinois, de faire des relectures incroyablement détaillées et ponctuelles. Nous sommes contents qu'elle puisse maintenant consacrer son temps à ses hobbies dont le vélo, la photo, les ordinateurs, la musique et le tennis

Quand **Steve Milano** n'est pas en train de coder en six langues différentes à son travail, de rédiger une excellente critique de SQL — Tête la première ou de jouer du punk rock avec son groupe Onion Flavored Rings dans des sous-sols non-ventilés à travers le pays, on peut le trouver chez lui avec ses chats Ralph et Squeak.

« Shelley » Moira Michelle Rheams, Master d'éducation, MCP, MCSE, enseigne et dirige le Early Childhood Education Program au Delgado Community College à la Nouvelle Orléans (West Bank Campus). En ce moment elle s'amuse à mettre des cours en ligne pour répondre aux besoins de la communauté changeante de la Nouvelle Orléans post-Katrina, et nous la remercions d'avoir pu nous faire entrer dans son emploi du temps surchargé.

Jamie Henderson est un cadre architecte de systèmes aux cheveux violets et occupant son temps libre entre le violoncelle, la lecture, les jeux vidéo et les films en DVD.

Cette équipe fantastique est la raison pour laquelle le code est les exercices de ce livre font réellement ce qu'ils sont censés faire et grâce à quoi, quand vous aurez fini ce livre, vous serez un programmeur SQL assuré. Leur attention aux détails nous a aussi empêché d'être trop malin ou trop condescendant, et même, parfois, trop bizarre.

Remerciements

Mes éditeurs :

En premier lieu, je voudrais remercier mon éditeur, **Brett McLaughlin**, pour non pas un, mais deux camps d'entraînement Tête la première. Brett a été plus qu'un éditeur. Il a été quelque chose entre un sherpa et un testeur d'idées. Ce livre n'aurait jamais pu être écrit sans ses conseils, son soutien et son intérêt. Il m'a non seulement comprise dès la première audition, mais son appréciation de mon humour parfois un peu surréel a fait de ce livre la meilleure expérience d'écriture que j'ai jamais eue. Il m'a donné tout un tas de conseils, d'indices et et a été mon coach pendant tout ce temps. Merci, Brett !

Brett McLaughlin

Catherine Nolan a maintenant un énorme ulcère, à cause d'une malchance incroyable que j'ai eu vers la fin du processus d'édition. C'est grâce à elle si le livre n'est pas sorti en 2008 et peut-être s'il existe tout simplement. La fin était un peu comme de jongler avec des châtons et elle n'en a jamais laissé tomber un seul. J'avais grand besoin d'un échéancier et Catherine est la meilleure organisatrice que j'ai jamais rencontré. Et je pense que j'ai été son plus grand défi jusqu'ici. Espérons que son prochain projet se déroule sans problème, elle l'a plus que mérité.

Catherine Nolan

L'équipe O'Reilly :

La responsable de conception **Louise Barr** a été à la fois une grande amie et une conceptrice graphique d'exception. Elle a été capable d'acheminer mes idées bizarres vers des œuvres d'art impressionnantes qui rendent très clairs les concepts difficiles. Toute la composition est d'elle, et je sais déjà qu'à de nombreuses reprises dans ce livre vous voudrez la remercier vous aussi.

Lou Barr

Mais nous aurions imprimé beaucoup d'erreurs sans la processus de relecture de l'équipe technique, et **Sanders Kleinfeld** a fait un travail fantastique d'éditeur de production, pour préparer le livre à l'impression. Il est aussi allé bien au-delà de son devoir, indiquant quelques gouffres conceptuels qui avaient vraiment besoin d'être comblés. Merci, Sanders !

Enfin, je veux remercier **Kathy Sierra** et **Bert Bates** pour la création de cette merveilleuse série et pour la meilleure et la plus stimulante intellectuellement de toutes les formations au camp d'entraînement Tête la première. Sans ces trois jours, hé bien, je ne veux même pas imaginer à quel point il aurait été difficile de suivre le style Tête la première. Et les derniers commentaires éditoriaux de Bert ont été douloureusement justes et ont grandement amélioré ce livre.

L'équipe française

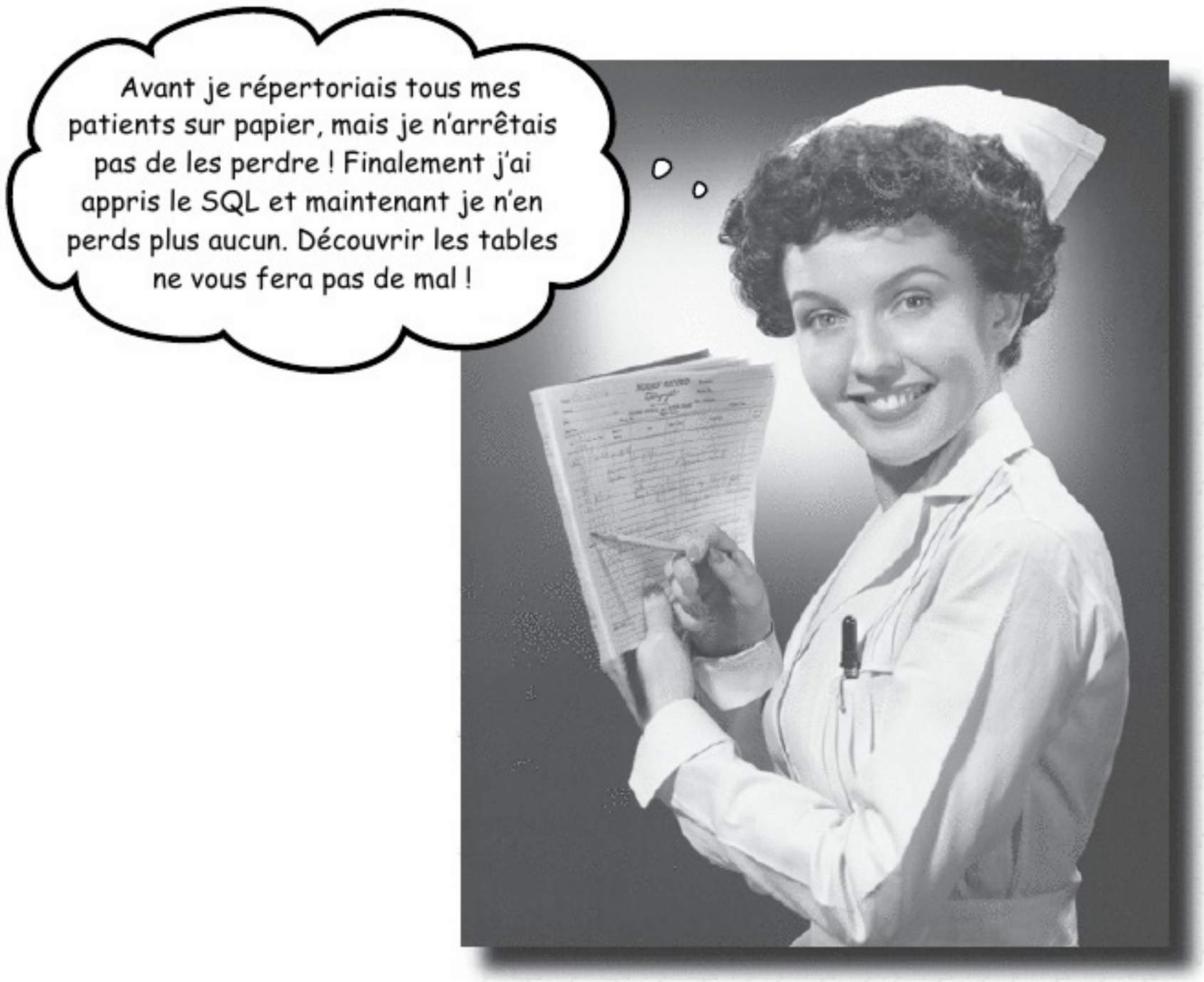
Sophie Govaere a traduit cet ouvrage.

Sophie est spécialisée en langues et en pédagogie. Elle s'intéresse beaucoup à la psychologie cognitive et à l'informatique de façon générale.

Olivier Fillol a relu la traduction.

Olivier est Ingénieur de formation. Il s'est d'emblée orienté vers les activités Système / DBA Oracle sur différents projets en environnement Télécoms. Il a pu ainsi appréhender l'ensemble des contraintes relatives aux environnements de production critiques : disponibilité, sécurité, scalabilité, supervision. Actuellement, Olivier intervient sur des missions dans le domaine de l'informatique décisionnelle.

Chaque chose à sa place

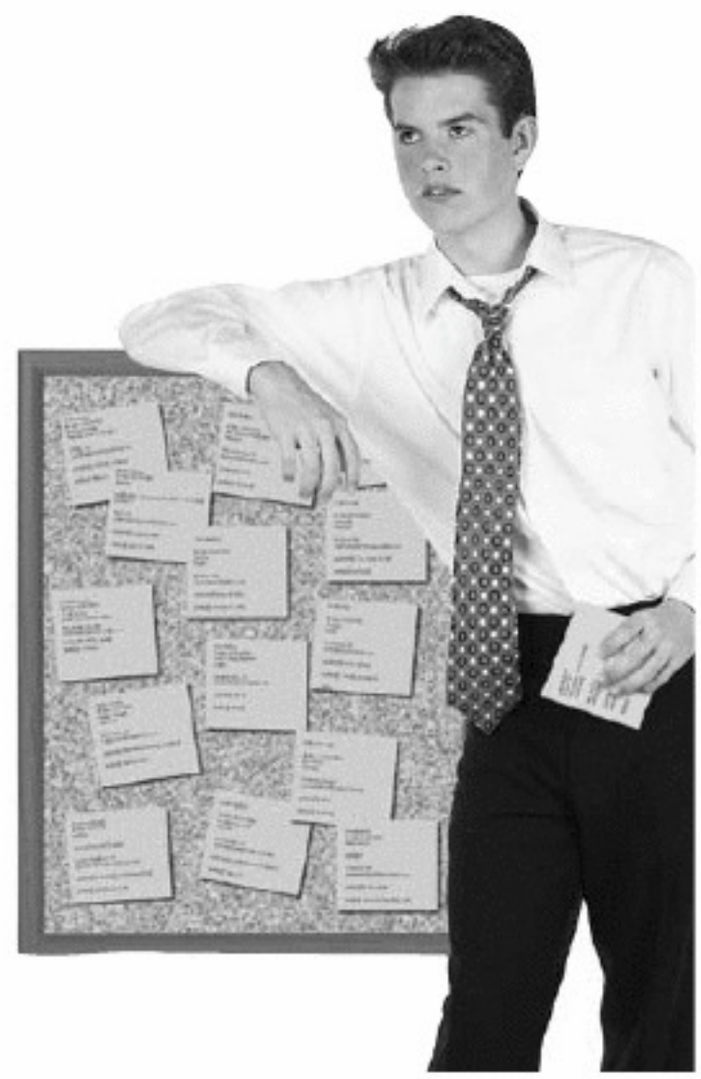


Vous ne trouvez pas énervant de perdre quelque chose ? Que ce soient vos clés de voiture, ce bon de réduction de 25% chez Ikéa ou les données de votre application, il n'y a rien de pire que de ne pas pouvoir **trouver ce dont on a besoin**... au moment où on en a besoin. Pour vos applications, il n'existe pas de meilleur endroit qu'une **table** pour conserver vos informations importantes. Alors tournez la page, entrez et venez vous promener dans le monde des **bases de données relationnelles**.

Définir vos données

Greg connaît beaucoup de personnes célibataires en mal d'amour. Il aime savoir où en sont ses amis et adore les présenter les uns aux autres. Il a beaucoup d'informations sur eux griffonnées sur des pense-bêtes comme celui-ci :

Cela fait très longtemps que Greg utilise ce système. Depuis la semaine dernière il développe son réseau en ajoutant des gens qui cherchent un nouveau travail et ses listes s'allongent rapidement. Très rapidement…



Quelques-uns des pense-bêtes de Greg

MUSCLEZ VOS NEURONES

Existe-t-il une meilleure façon d'organiser ces informations ? Que feriez-vous ?

Tout à fait. C'est exactement ce qu'il nous faut : une base de données.

Mais avant de commencer à créer des bases de données, vous devez connaître un peu mieux les *genres* de données que vous voulez conserver et les différentes façons de les *catégoriser*.

À vos crayons !

Voici quelques-uns des pense-bêtes de Greg. Cherchez les points communs entre les informations collectées par Greg sur chaque personne. Donnez à chaque donnée commune une étiquette décrivant sa catégorie d'information, puis notez-la dans les espaces ci-dessous.

Anne Neveu
Anniv : 7/1/1962
Ingénieur informatique

Célibataire, mais a un copain

Grenoble, 38
anne@planchesagogo.com

Centres d'intérêt : collection de livres, fabrication de la bière, équitation

Cherche : Nouveau boulot

Étienne Gros
Anniv : 10/9/1964
Administrateur Système

Célibataire

Nice, 06
nessayepas@pizzaprestissima.com

Centres d'intérêt : randonnée, écriture,

Cherche : amis, rencontres avec des femmes

Alain Souk
Anniv : 1/7/1966
Ingénieur aéronautique

Marié

Orléans, 45
Souk@pizzaprestissima.com

Centres d'intérêt : RPG, programmation

Cherche : Rien

Lucie Mendoza
Anniv : 19/8/1979
Admin Système Unix

Mariée

Nice, 06
lucie@cafestarbuzz.com

Centres d'intérêt : théâtre, danse

Cherche : un nouveau boulot

Cherche

. .....

. .....

. .....

. .....

. .....

. .....

. .....

. .....

. .....

. .....

À vos crayons ! Solution

Voici quelques-uns des pense-bêtes de Greg. Cherchez les points communs entre les informations collectées par Greg sur chaque personne. Donnez à chaque donnée commune une étiquette décrivant sa catégorie d'information, puis notez-la dans les espaces ci-dessous.

Maintenant que nous avons créé ces catégories, nous pouvons les utiliser pour organiser nos données.

Prénom

Anne Neveu
Anniv : 7/1/1962
Ingénieur informatique

Statut → Célibataire mais a un copain

Grenoble, 38
anne@planchesagogo.com

Centres d'intérêt : collection de livres, fabrication de la bière, équitation

Cherche: Nouveau boulot

Nom

Étienne Gros
Anniv : 10/9/1964
Administrateur système

Célibataire

Nice, 06
nessayepas@pizzaprestissima.com

Centres d'intérêt : randonnée, écriture

Cherche : amis, rencontres avec des femmes

Nous avons divisé les noms en prénoms et noms de famille. Cela vous aidera à trier les données plus tard.

Prénom .....

Nom .....

Anniversaire .....

Profession .....

Statut .....

Lieu .....

Email .....

Centres d'intérêt .....

Cherche .....

Alain Souk
Anniv : 1/7/1966
Profession → Ingénieur aéronautique

Marié

Lieu → Orléans, 45
Email → souk@pizzaprestissima.com

Centres d'intérêt : RPG, programmation

Cherche : Rien

Lucie Mendoza
Anniv : 19/8/1979
Admin système Unix

Mariée

Nice, 06
lucie@cafestarbuzz.com

Centres d'intérêt : théâtre, danse

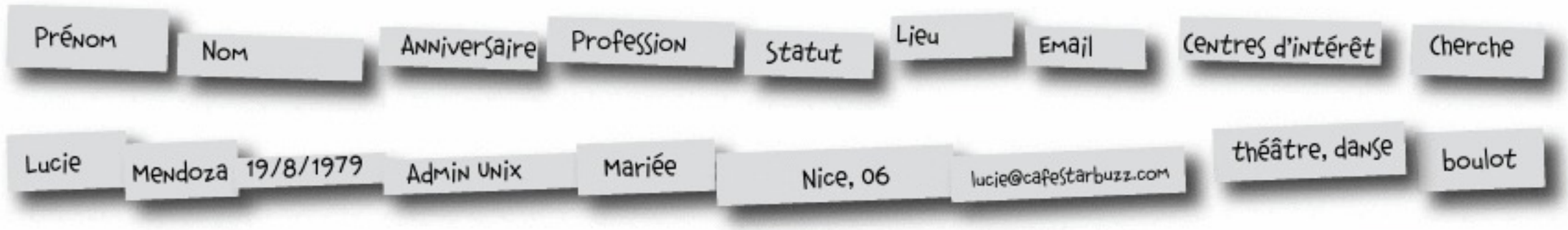
Cherche : un nouveau boulot

Greg a déjà utilisé les noms de catégories « Anniv », « Centres d'intérêt » et « Cherche » pour certaines informations sur ses pense-bêtes.

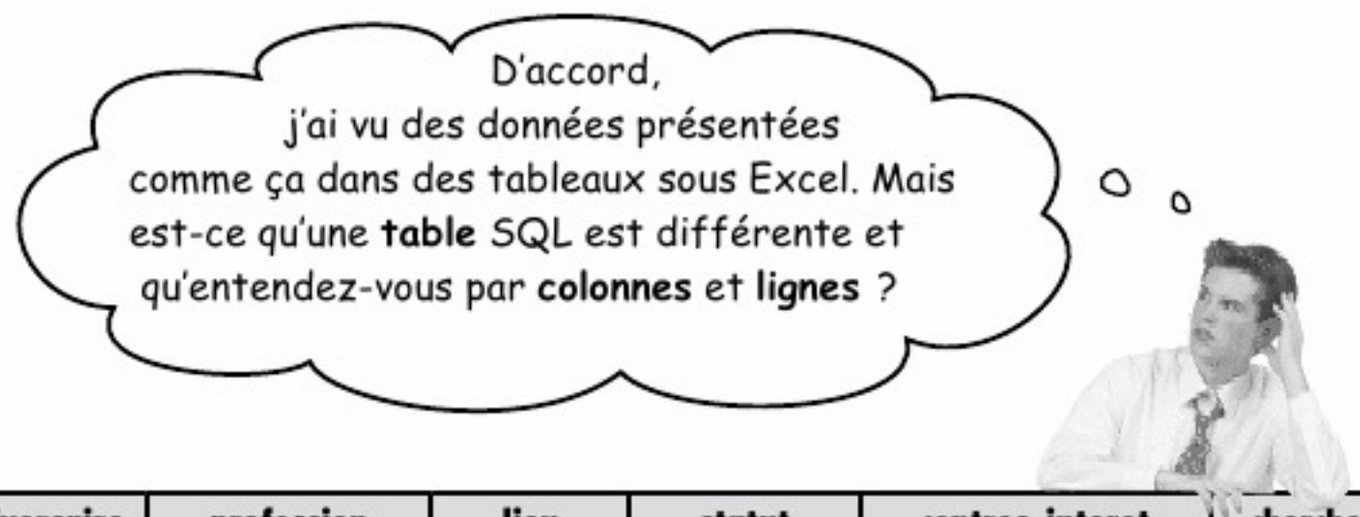
Regardez vos données triées par catégories

Regardons vos données d'une façon différente. Si vous découpez chaque pense-bête en morceaux, puis disposez les morceaux horizontalement, vous obtenez quelque chose qui ressemble à ceci :

Puis si vous découpez un autre pense-bête avec les catégories que vous venez d'identifier et que vous mettez les morceaux au-dessus des informations correspondantes, vous obtenez quelque chose qui ressemble beaucoup à ceci :



Voici ces mêmes informations joliment disposées dans une **TABLE** en **colonnes** et en **lignes.**



nom	prenom	email	anniversaire	profession	lieu	statut	centres_interet	cherche
Neveu	Anne	anne@planchesagogo.com	7-1-1962	Ingénieur informatique	Grenoble, 38	Célibataire mais a un copain	Collection de livres, fabrication de la bière, équitation	Nouveau boulot
Gros	Étienne	nessayepas@pizzaprestissima.com	10-9-1964	Administrateur système	Nice, 06	Célibataire	randonnée, écriture	amis, rdv avec des femmes
Souk	Alain	souk@pizzaprestissima.com	1-7-1966	Ingénieur aéronautique	Orléans, 45	Marié	RPG, programmation	Rien
Mendoza	Lucie	lucie@cafestarbuzz.com	19-8-1979	Administratrice système Unix	Nice, 06	Mariée	théâtre, danse	Nouveau boulot

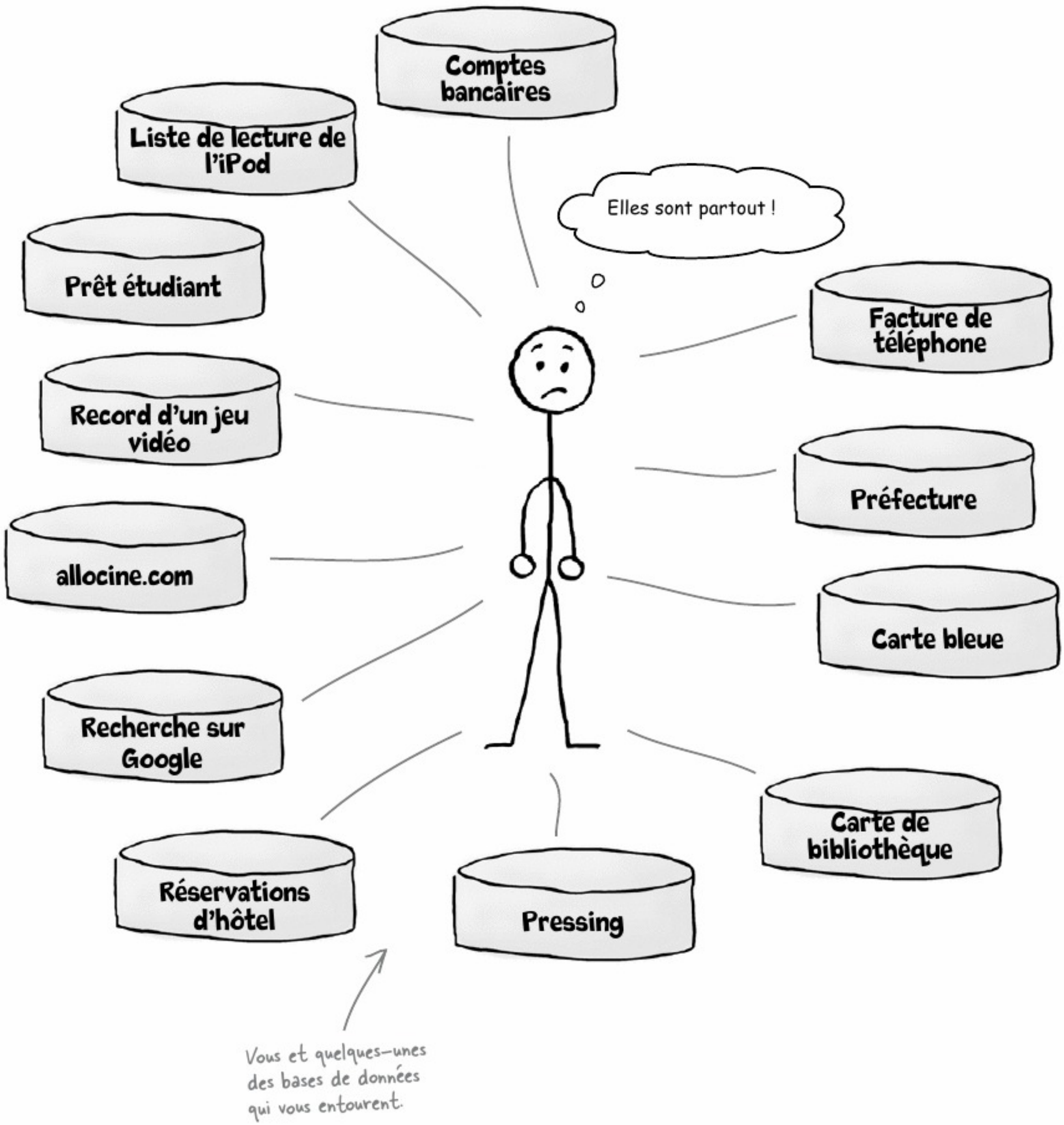
Qu'y a-t-il dans une base de données ?

Avant de rentrer dans les détails de ce que sont les tables, les lignes et les colonnes, reculons d'un pas pour avoir une vue d'ensemble. La première structure SQL que vous avez besoin de connaître est le récipient qui contient toutes vos tables, connu sous le nom de ***base de données***.

Une **base de données** est un récipient qui contient des tables et d'autres structures SQL liées à ces tables.

Chaque fois que vous faites une recherche sur Internet, que vous allez faire des courses, que vous appelez les renseignements, que vous utilisez votre disque dur, que vous faites une réservation, que vous recevez une amende pour excès de vitesse ou que vous achetez des provisions, une base de données reçoit une demande d'informations, que l'on appelle aussi ***requête***.

Dans les diagrammes et les schémas, les bases de données sont représentées par des cylindres. Donc, quand vous voyez ceci, pensez « base de données ».


Comptes bancaires
Liste de lecture de l'iPod
Elles sont partout !
Prêt étudiant
Facture de téléphone
Record d'un jeu vidéo
Préfecture
allocine.com
Carte bleue
Recherche sur Google
Carte de bibliothèque
Réservations d'hôtel
Pressing
Vous et quelques-unes des bases de données qui vous entourent.

Anatomie d'une base de données

Votre base de données vue à travers des lunettes à rayons x...

Pensez à une base de données comme à un récipient qui contient des informations...

Une table.

Voici les colonnes.

Une autre table.

Voici les lignes.

Une autre table.

Encore une autre table.

Une base de données contient des tables.

Une **table** est une structure à l'intérieur de votre base de données qui contient des données, organisées en **colonnes** et en **lignes**.

Vous souvenez-vous de ces catégories que vous avez identifiées ? Chaque catégorie devient une **colonne** dans votre table. Ces valeurs peuvent être dans la même colonne : célibataire, marié, divorcé.

Une **ligne** d'une table contient toutes les informations sur un objet de votre table. Dans la nouvelle table de Greg, une ligne serait toutes les données sur une personne. Voici un exemple de certaines données pouvant être sur une même ligne : Jean, Valjean, célibataire, écrivain, jv@planchesagogo.com.

> Les informations contenues dans la base de données sont organisées dans des **tables**.

Vous ÊTES la table

Ci-dessous, vous trouverez des pense-bêtes et une table. Votre mission est de vous mettre dans la peau de la table partiellement remplie et de compléter ce qu'il faut pour trouver la paix intérieure. Après avoir fait l'exercice, tournez la page pour voir si vous faites vraiment un avec la table.

La crêperie du port
7
24/4
manque de confiture
22h35
fourrée à la confiture

Café Starbuzz
23/4
fourrée à la confiture
9
7h43
presque parfaite

La crêperie du port
5
25/4
fourrée à la confiture
8h56
grasse

fourrée à la confiture
pas très chaude mais savoureuse
6
Le roi de la crêpe
26/4
21h39

Utilisez l'un des champs comme titre pour donner à la table un nom significatif.

boutique				
			9	
		25/4	5	
				manque de confiture

Vous ÊTES la table : Solution

Vous deviez devenir la table partiellement remplie et compléter ce qu'il fallait pour atteindre la paix intérieure.

Vous devriez avoir trouvé le titre de la table à partir des pense-bêtes.

crepes_confiture

boutique	heure	date	note	commentaires
Café Starbuzz	7h43	23/4	9	presque parfaite
La crêperie du port	8h56	25/4	5	grasse
Le roi de la crêpe	21h39	26/4	6	pas très chaude mais savoureuse
La crêperie du port	22h35	24/4	7	manque de confiture

Ne vous inquiétez pas si vos réponses pour les noms de colonnes ne correspondent pas exactement aux nôtres.

Les bases de données contiennent des données reliées entre elles

Toutes les tables d'une base de données doivent être ***liées*** d'une façon ou d'une autre. Par exemple, voici des tables qui pourraient se trouver dans une base de données contenant des informations sur des crêpes :

Voici une base de données contenant trois tables. La base de données s'appelle "mes_encas".

Les noms de bases de données et de tables sont normalement en minuscules.

mes_encas

Table contenant les informations sur les crêpes à la confiture.

crepes_confiture

boutique	heure	date	note	commentaires
Café Starbuzz	7h43	23/4	9	presque parfaite
La crêperie du port	8h56	25/4	5	grasse
Le roi de la crêpe	21h39	26/4	6	pas très chaude mais savoureuse

Table contenant les informations sur les crêpes au sucre.

crepes_sucre

boutique	heure	date	note	commentaires
Le roi de la crêpe	21h39	26/4	8	tiède mais pas chaud
Café Starbuzz	7h43	23/4	4	pas assez de sucre glace
La crêperie du port	8h56	25/4	6	grasse
La crêperie du port	22h35	24/4	7	pas frais

Table contenant les informations sur les encas autres que les crêpes.

autres_encas

boutique	heure	date	gateau	note	commentaires
Café Starbuzz	22h35	24/4	pain d'épice	6	trop épicé
Café Starbuzz	7h43	23/4	éclair	8	chocolat !
Le roi de la crêpe	21h39	26/4	tartelette	4	manque de fruits
La crêperie du port	8h56	25/4	croissant	9	chaud, croustillant

Les tables vues de plus près

Une **colonne** est une donnée conservée par votre table. Une **ligne** est un ensemble unique de colonnes qui décrit les attributs d'un objet particulier. Ensemble, les colonnes et les lignes forment une table.

Voici un exemple de ce à quoi peut ressembler une table de carnet d'adresses contenant vos informations personnelles. Vous verrez souvent le mot champ utilisé à la place de colonne. Ils désignent la même chose. De même, ligne et enregistrement sont souvent interchangeables.

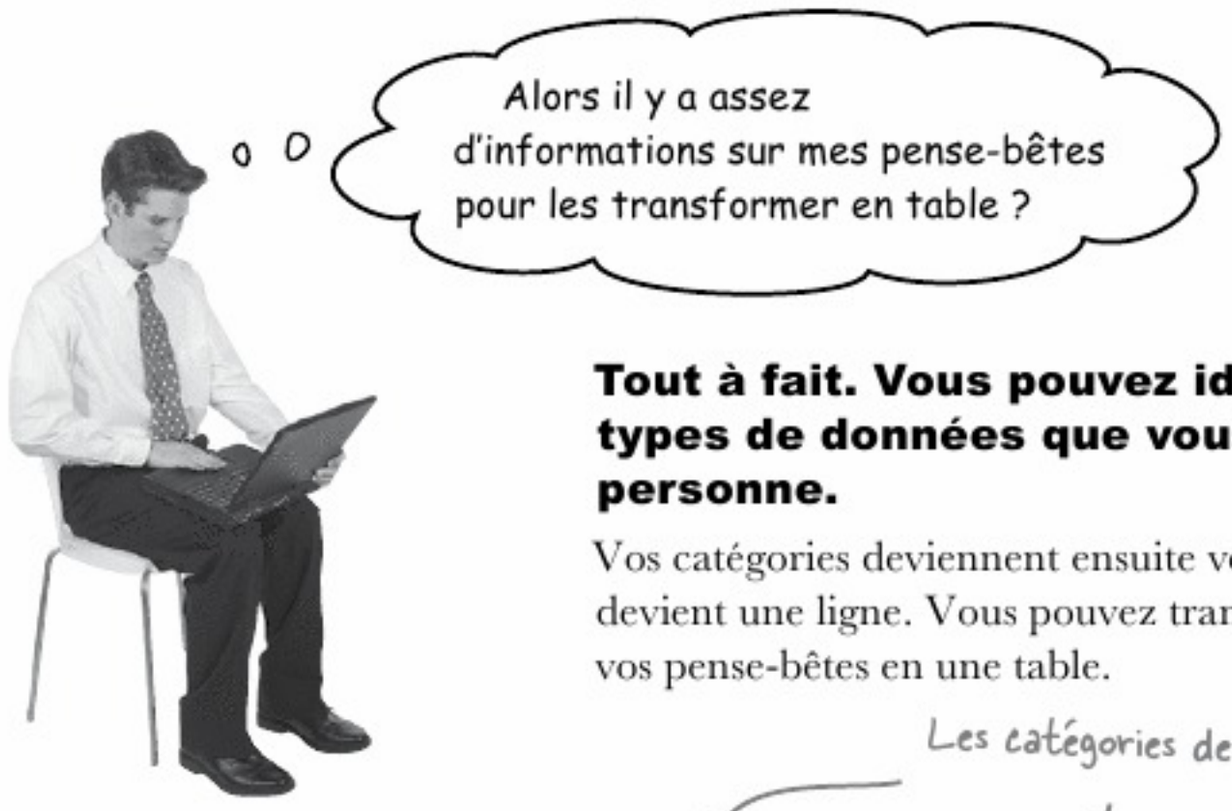
Voici les colonnes.

prenom	nom	adresse	ville	code_postal	num_id
Auguste	Préau	donnée	donnée	donnée	donnée
Albert	Durand	donnée	donnée	donnée	donnée
Marie	Herbier	donnée	donnée	donnée	donnée
Louis	Blanc	donnée	donnée	donnée	donnée

Voici les lignes.

Assemblez les colonnes et les lignes et vous obtenez une table.

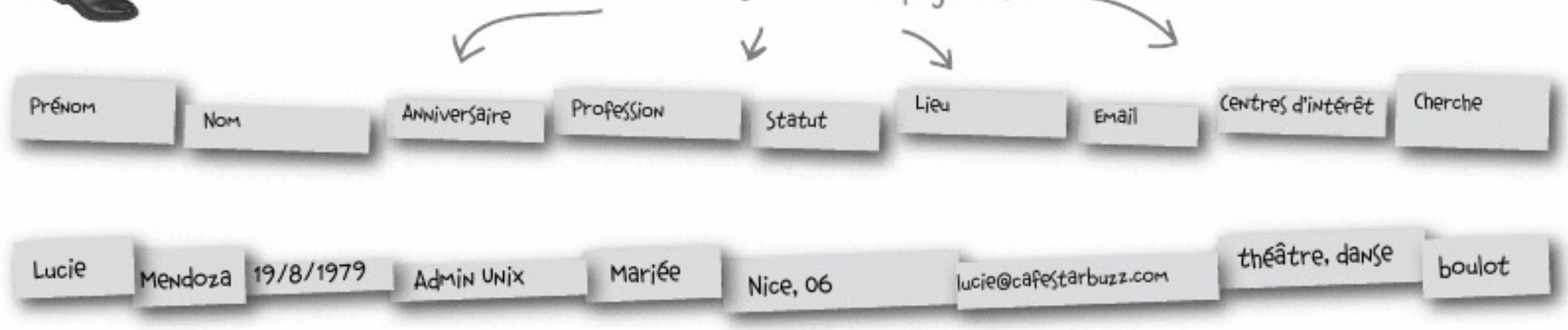
prenom	nom	adresse	ville	code_postal	num_id
Auguste	Préau	donnée	donnée	donnée	donnée
Albert	Durand	donnée	donnée	donnée	donnée
Marie	Herbier	donnée	donnée	donnée	donnée
Louis	Blanc	donnée	donnée	donnée	donnée



Tout à fait. Vous pouvez identifier les catégories des types de données que vous rassemblez pour chaque personne.

Vos catégories deviennent ensuite vos colonnes. Chaque pense-bête devient une ligne. Vous pouvez transformer toutes les informations de vos pense-bêtes en une table.

Les catégories de la page 7.



Données d'un pense-bête étalées pour former une ligne.

Maintenant vous savez que les catégories s'appellent des colonnes.

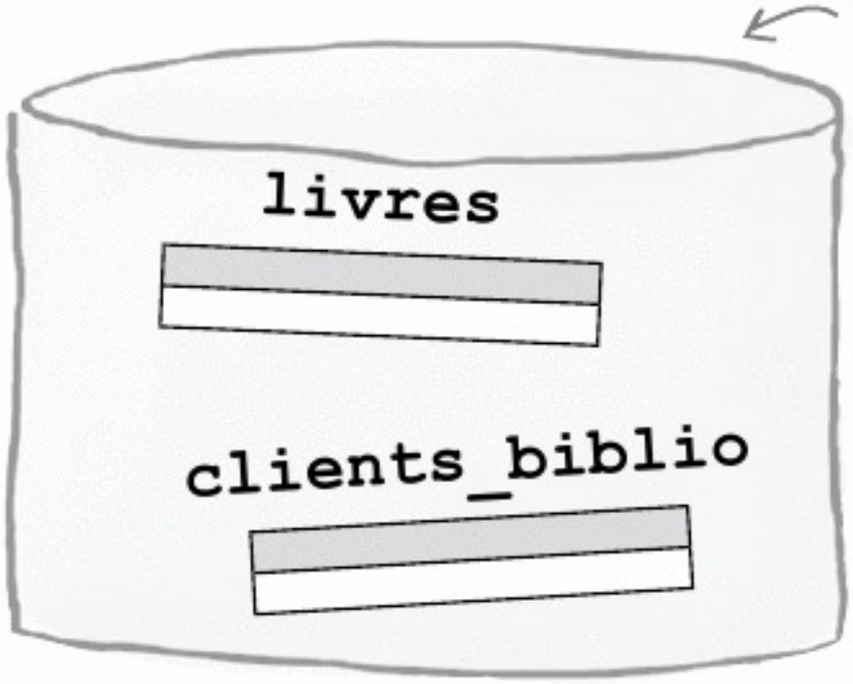
nom	prenom	email	anniversaire	profession	lieu	statut	centres_interet	cherche
Neveu	Anne	anne@ planchesagogo.com	7-1-1962	Ingénieur informatique	Grenoble, 38	Célibataire mais a un copain	Collection de livres, fabrication de la bière, équitation	Nouveau boulot
Gros	Étienne	nessayepas@ pizzaprestissima. com	10-9-1964	Administrateur système	Nice, 06	Célibataire	randonnée, écriture	amis, rdv avec des femmes
Souk	Alain	souk@ pizzaprestissima. com	1-7-1966	Ingénieur aéronautique	Orléans, 45	Marié	RPG, programmation	Rien
Mendoza	Lucie	lucie@ cafestarbuzz .com	19-8-1979	Administratrice système Unix	Nice, 06	Mariée	théâtre, danse	Nouveau boulot

...et que les données de chaque pense-bête peuvent être disposées sur une ligne appelée un enregistrement.

Observez les bases de données et les tables ci-dessous. Pensez aux catégories de données que vous pouvez trouver dans chacune d'elles et trouvez des colonnes possibles pour chaque table.

bdd_bibliotheque

Base de données pour une bibliothèque.

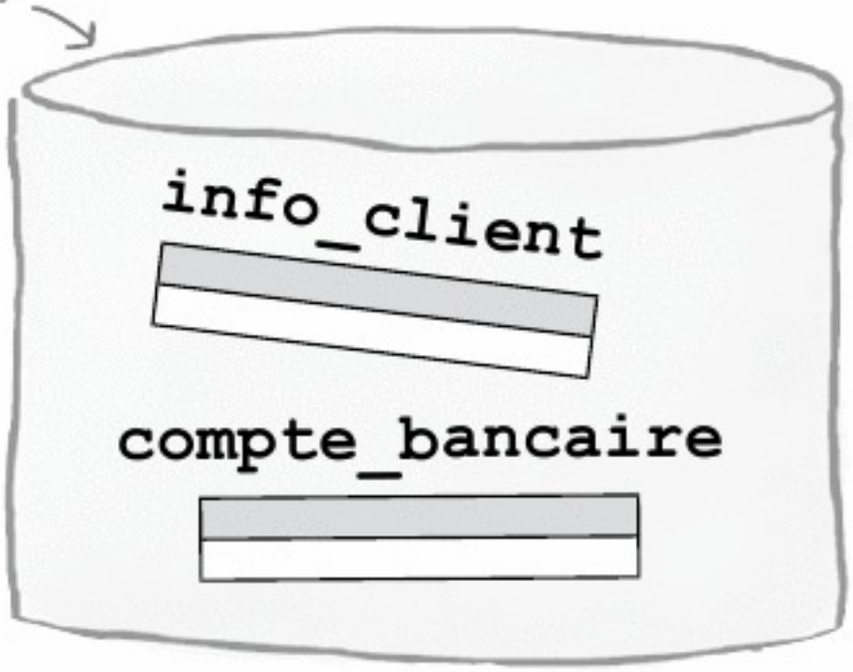


livres : .....

clients_biblio : .....

bdd_banque

Base de données pour une banque.

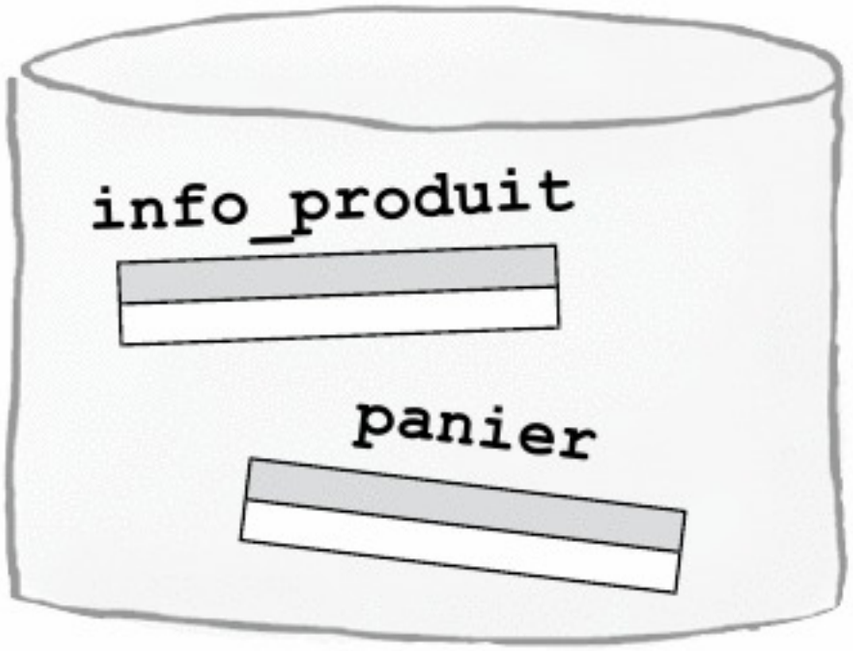


info_client : .....

compte_bancaire : .....

bdd_magasinenligne

Base de données pour un magasin en ligne.



info_produit: .....

panier: .....

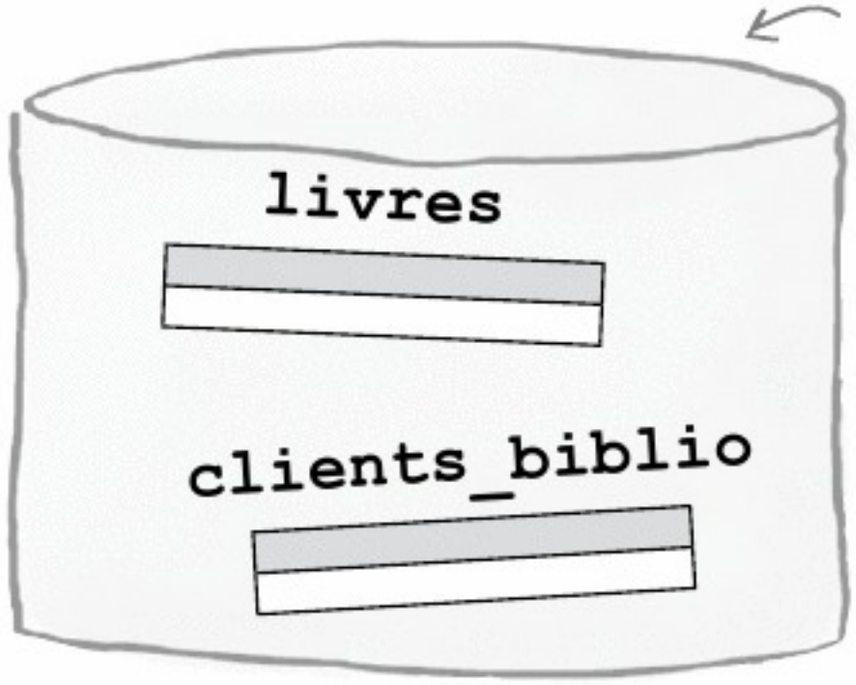
Solutions des exercices

Observez les bases de données et les tables ci-dessous. Pensez aux catégories de données que vous pouvez trouver dans chacune d'elles. Trouvez des colonnes possibles pour chaque table.

bdd_bibliotheque

Base de données pour une bibliothèque.

Ne vous inquiétez pas si vos réponses pour les noms des colonnes ne correspondent pas exactement aux nôtres.

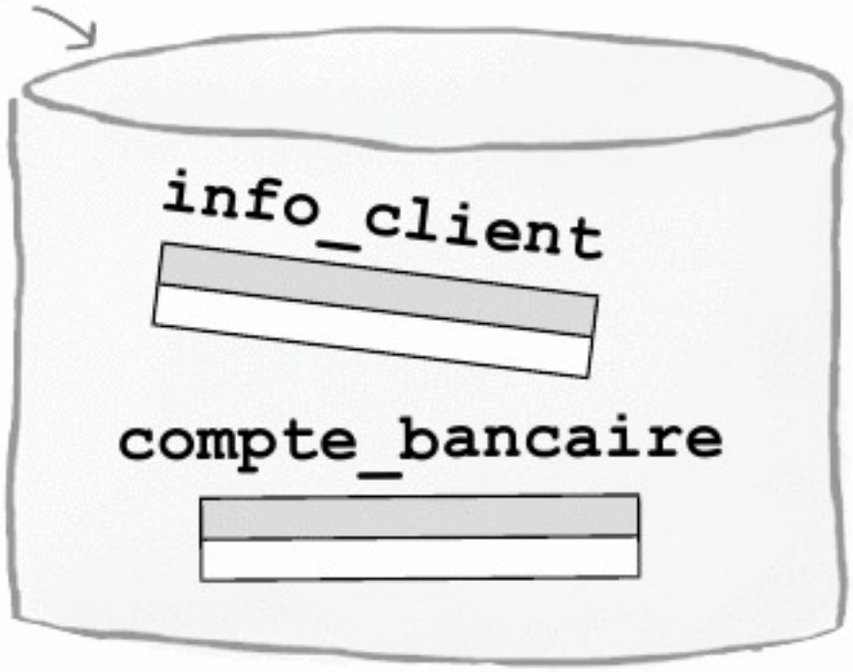


livres : titre, auteur, editeur, isbn

clients_biblio : prenom, nom, adresse

bdd_banque

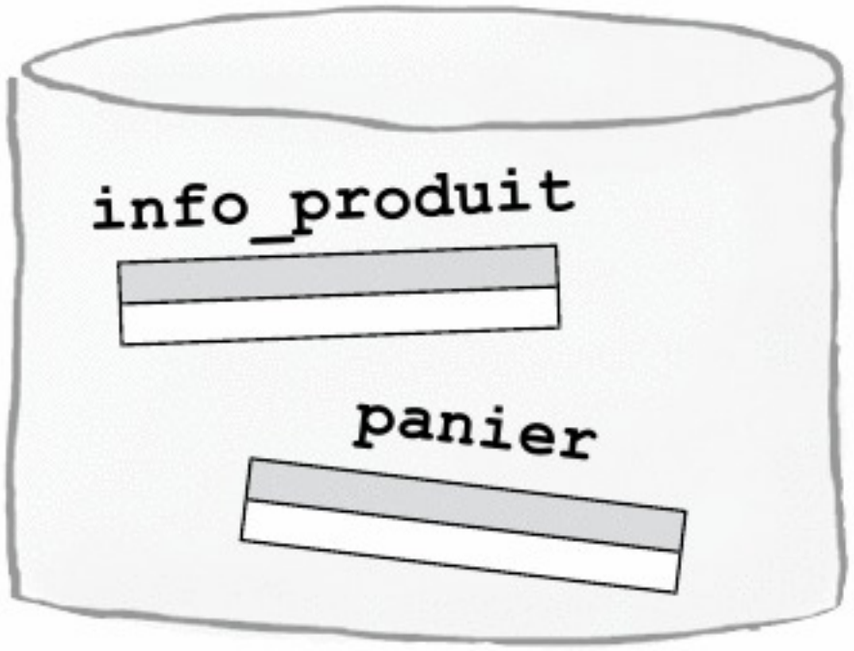
Base de données pour une banque.



info_client : prenom, nom, adresse, numero_compte, numero_secu

compte_bancaire : solde, depots, retraits

bdd_magasinenligne



Base de données pour un magasin en ligne.

info_produit : nom, taille, prix

panier : montant_facture, id_client

Prenez les commandes !

Lancez votre Système de Gestion de Base de Données Relationnelle SQL (SGBDR) et ouvrez une fenêtre de ligne de commande ou un environnement graphique qui vous permette de communiquer avec votre SGBDR. Voici la fenêtre de notre terminal après que nous ayons lancé MySQL.

```
Fichier Edition Fenêtre Aide DonneMoiUnOrdreBaby
Welcome to the SQL monitor. Commands end with ; or \g.

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

>
```

Ce signe est l'invite de commande. Vous devez entrer vos commandes juste après.

Pour commencer vous allez devoir créer une base de données pour y mettre toutes vos tables.

Les espaces ne sont pas autorisés dans les noms de bases de données et de tables en SQL, on peut donc utiliser un tiret bas à la place.

1 Entrez la ligne de code ci-dessous pour créer une base de données appelée **`liste_greg`**.

```
CREATE DATABASE liste_greg;
```

CREATE DATABASE est la commande.

Le nom de la base de données est liste_greg

Votre commande doit se finir par un point-virgule.

```
Fichier Edition Fenêtre Aide DonneMoiUnOrdreBaby
> CREATE DATABASE liste_greg;
Query OK, 1 row affected (0.01 sec)
```

Voici la réponse du SGBDR vous disant que votre requête a été exécutée avec succès.

Ouvrez les yeux !

Avez-vous lu l'intro ?

Nous utilisons MySQL pour gérer nos bases de données. Les commandes de votre Système de Gestion de Base de Données (SGBD) peuvent donc être légèrement différentes. Reportez-vous à l'annexe B pour les instructions d'installation de MySQL sur votre serveur.

Vous avez maintenant besoin de demander au SGBDR d'*utiliser*, de fait, la base de données que vous venez de créer :

```
USE liste_greg;
```

Maintenant tout ce que nous ferons se passera à l'intérieur de la base de données liste_greg !

```
Fichier Edition Fenêtre Aide UtiliSE
> USE liste_greg;
Database changed
```

il n'y a pas de

Questions stupides

Q: Pourquoi dois-je créer une base de données si je n'ai qu'une table ?

R: Dans le langage SQL, toutes les tables doivent se trouver à l'intérieur de bases de données. Il y a de bonnes raisons pour cela. L'une des caractéristiques de SQL est sa capacité à contrôler l'accès de différents utilisateurs à vos tables. Pouvoir autoriser ou refuser l'accès à une base de données toute entière est parfois plus simple que de devoir contrôler les permissions pour toutes les tables.

Q: J'ai remarqué que nous avons entré la commande CREATE DATABASE entièrement en majuscules. Est-ce obligatoire ?

R: Certains systèmes nécessitent que des mots-clés soient entrés en majuscules mais SQL est insensible à la casse. Cela signifie qu'il n'est pas nécessaire d'utiliser des majuscules pour les commandes. Cependant on considère cela comme une bonne habitude de programmation en SQL. Prenez la commande que nous venons de taper :

```
CREATE DATABASE liste_greg;
```

Les majuscules permettent de faire facilement la différence entre la commande (`CREATE DATABASE`) et le nom de la base de données (`liste_greg`).

Q: Y a-t-il quelque chose que je dois savoir pour nommer mes bases de données, mes tables et mes colonnes ?

R: De manière générale, il vaut mieux créer des noms descriptifs. Cela implique parfois que vous ayez besoin d'utiliser plus d'un mot dans un nom. Vous ne pouvez pas utiliser d'espaces dans vos noms, donc le tiret bas vous permet de créer des noms plus descriptifs. Voici des variantes que vous pouvez voir utilisées :

```
liste_greg
listegreg
Listegreg
listeGreg
```

En général il vaut mieux éviter de mettre vos noms en majuscules pour éviter les confusions puisque SQL est insensible à la casse.

Q: Et si je préfère utiliser « listeGreg » sans tiret bas ?

R: Allez-y. L'important est d'être logique. Si vous utilisez `listeGreg` comme nom de base de données sans tiret bas et avec une majuscule au deuxième mot, pour être cohérent, vous devez conserver cette convention de nommage pour toutes les tables de cette base de données, en appelant, par exemple, votre table `mesContacts`.

Q: Peut-on utiliser des apostrophes, des accents ?

R: L'apostrophe est réservée à une autre utilisation en SQL. Il existe des façons qui rendent possible de l'utiliser, mais il est beaucoup plus facile de l'omettre. De même, les accents ne sont pas pris en charge dans les noms de bases de données, de tables ou de colonnes.

Q: J'ai remarqué un point-virgule à la fin de la commande CREATE DATABASE également. Pourquoi en avons-nous besoin ?

R: Le point-virgule sert à indiquer la fin de la commande.

Les majuscules et le tiret bas vous aident à programmer en SQL (même si SQL n'a pas besoin d'eux !)

Dresser la table : l'instruction CREATE TABLE

Voyons tout ceci à l'œuvre avec les données sur les crêpes. Supposons que vous ayez des difficultés à vous souvenir du type de crêpe de votre liste d'encas à partir de son nom, vous pouvez ***créer une table*** pour ne pas avoir à vous en souvenir. Voici une commande unique à entrer dans la fenêtre de votre console. Quand vous l'avez tapée, vous pouvez appuyer sur ENTRÉE pour demander à votre SGBDR SQL d'exécuter la commande.

`liste_crepes`

nom_crepe	type_crepe
Miel amande	fourrée
Suzette	flambée
Citron pressé	sirop
Sirop d'érable	sirop
Crème de marron	fourrée

Voici la commande SQL pour créer la table. Remarquez les majuscules.

Le nom de votre table doit être en minuscules et avoir des tirets bas à la place des espaces.

Pressez simplement entrée pour commencer une nouvelle ligne de votre commande et faciliter la lecture.

```
CREATE TABLE liste_crepes
(
  nom_crepe VARCHAR(15),
  type_crepe VARCHAR(7)
);
```

La parenthèse ouvrante ouvre la liste des colonnes à créer.

Le nom de la première colonne de la table.

La virgule sépare les colonnes qui sont créées.

Le nom de la deuxième colonne.

La parenthèse fermante ferme la liste des colonnes.

Le point-virgule signale au SGBDR SQL qu'il a atteint la fin de la commande.

Ceci est un TYPE de DONNÉES. Il correspond à VARiable CHARacter (chaîne variable) et s'utilise pour conserver des informations sous forme de texte. Le (7) signifie que le texte qu'il contient peut avoir jusqu'à 7 caractères.

Créer une table plus compliquée

Vous souvenez-vous des colonnes de la table de Greg ? Nous les avons griffonnées sur un pense-bête. Vous en aurez besoin pour écrire votre commande **CREATE TABLE**.

Vous allez utiliser la commande CREATE TABLE pour passer de ceci...

...à cela

nom	prenom	email	anniversaire	profession	lieu	statut	centres_interet	cherche

MUSCLEZ VOS NEURONES

Les noms de colonnes du pense-bête diffèrent de ceux de la table ci-dessus de deux façons. Lesquelles et pourquoi sont-elles significatives ?

Voyez comme SQL est facile à écrire

Vous avez vu que, pour créer une table, vous organisez vos données par catégories, en colonnes. Puis vous trouvez le bon type de données et la largeur de chaque colonne. Après avoir estimé la largeur que doit avoir chaque colonne, écrire le code coule de source.

À vos crayons !

Le code situé à gauche est notre instruction CREATE TABLE pour la nouvelle base de données de Greg. Essayez de deviner ce que fait chaque ligne de la commande CREATE TABLE. Ajoutez également un exemple de donnée pour chaque colonne.

```
CREATE TABLE mes_contacts
(
   nom VARCHAR(30),
   prenom VARCHAR(20),
   email VARCHAR(50),
   anniversaire DATE,
   profession VARCHAR(50),
   lieu VARCHAR(50),
   statut VARCHAR(20),
   centres_interet VARCHAR(100),
   cherche VARCHAR(100)
);
```


```
CREATE TABLE mes_contacts

(

    nom VARCHAR(30),

    prenom VARCHAR(20),

    email VARCHAR(50),

    anniversaire DATE,

    profession VARCHAR(50),

    lieu VARCHAR(50),

    statut VARCHAR(20),

    centres_interet VARCHAR(100),

    cherche VARCHAR(100)

);
```

Voici ce que fait chaque ligne de la commande CREATE TABLE et des données en exemple pour chaque type de colonne.

Crée une table appelée « mes_contacts »	
Ouvre la liste des colonnes à ajouter	
Ajoute une colonne appelée « nom » qui peut contenir jusqu'à 30 caractères	« Bouquet »
Ajoute une colonne appelée « prenom » qui peut contenir jusqu'à 20 caractères	« Carole »
Ajoute une colonne appelée « email » qui peut contenir jusqu'à 50 caractères	« carole_bouquet@pizzaprestissima.com »
Ajoute une colonne appelée « anniversaire » qui peut contenir une date sous la forme AAAA-MM-JJ	« 1980-09-05 »
Ajoute une colonne appelée « profession » qui peut contenir jusqu'à 50 caractères	« Écrivain technique »
Ajoute une colonne appelée « lieu » qui peut contenir jusqu'à 50 caractères	« Nice, 06 »
Ajoute une colonne appelée « statut » qui peut contenir jusqu'à 20 caractères	« Célibataire »
Ajoute une colonne appelée « centres_interet » qui peut contenir jusqu'à 100 caractères	« kayak, reptiles »
Ajoute une colonne appelée « cherche » qui peut contenir jusqu'à 100 caractères	« relations, amis »
Ferme la liste de colonnes à ajouter et le point-virgule met fin à la commande	

Créer la table mes_contacts, enfin

Maintenant que vous savez exactement ce que fait chaque ligne, vous pouvez entrer la commande **CREATE TABLE**. Vous pouvez taper une ligne à la fois en copiant le code du haut de cette page.

Ou vous pouvez tout entrer en une seule très longue ligne :

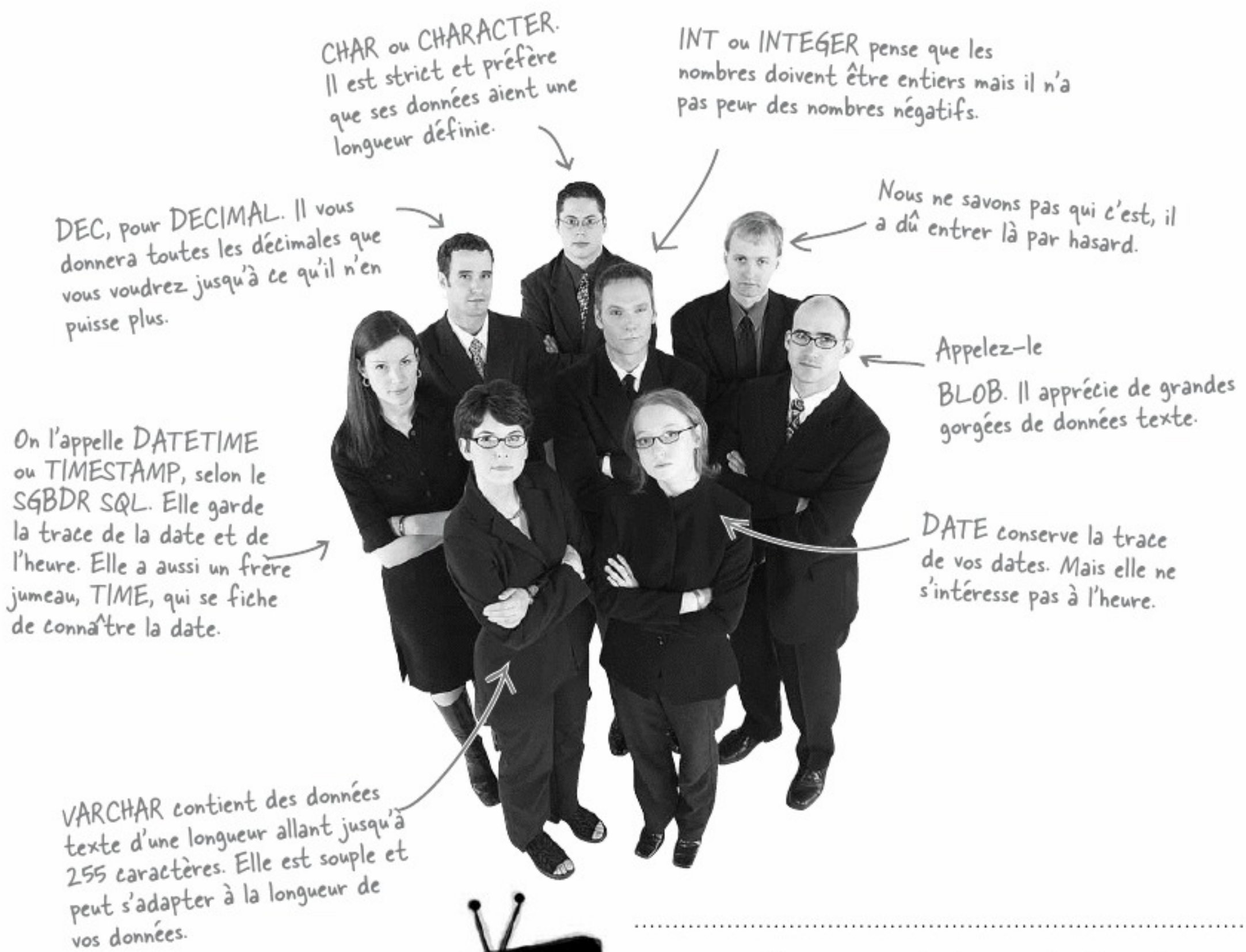
```
CREATE TABLE mes_contacts (nom VARCHAR(30), prenom VARCHAR(20), email VARCHAR(50), anniversaire DATE, profession VARCHAR(50), lieu VARCHAR(50), statut VARCHAR(20), centres_interet VARCHAR(100), cherche VARCHAR(100));
```

Quelle que soit la façon dont vous choisissez de l'entrer, assurez-vous de n'avoir oublié aucun caractère avant de presser entrée après le point-virgule : `centres_interet VARCHAR(100)` est une colonne très différente de `centresinteret VARCHAR(10)` !

Faites-nous confiance, ceci est réellement la commande, elle est juste écrite ex-trê-me-ment petit pour tenir sur la page !

Une réunion avec quelques types de données

Voici quelques-uns des types de données les plus utiles. Leur travail est de conserver vos données pour vous sans les abîmer. Vous avez déjà rencontré `VARCHAR` et `DATE`, mais saluez ceux-ci.



Ouvrez les yeux !

Ces noms de types de données peuvent ne pas fonctionner avec votre SGBDR SQL !

Malheureusement, il n'y a pas de noms universellement acceptés pour plusieurs de ces types de données. Votre SGBDR SQL personnel peut utiliser des noms différents pour un ou plusieurs de ces types. Consultez votre documentation pour trouver les noms corrects pour votre SGBDR.

QUEL TYPE DE DONNÉES ?

Déterminez quel est le type de données le plus logique pour chaque colonne. Pendant que vous y êtes, complétez les autres infos manquantes.

Ces deux nombres montrent combien de chiffres la base de données doit attendre avant et après la virgule.

Nom de colonne	Description	Exemple	Meilleur choix de type de données
prix	Le prix d'un produit à vendre	5678.39	DEC(6,2)
code_postal			
poids_atomique	Poids atomique d'un élément avec jusqu'à 6 décimales		
commentaires	Gros bloc de texte de plus de 255 caractères	Auguste, je suis à la réunion des actionnaires. Ils viennent de faire une démo et il y avait des petits canards en plastiques qui volaient à l'écran. Est-ce que c'est ton sens de l'humour ? Tu devrais aller faire un tour sur Monster.com.	
quantite	Nombre d'exemplaires de ce produit en stock		
tva		19.6	
titre_livre		SQL Tête la première	
sexe	Un caractère, soit H soit F		CHAR(1)
numero_tel	Dix chiffres sans ponctuation	0505552367	
initiales	Abréviation de deux caractères pour des initiales	TX, CA	
anniversaire		11/22/2006	DATE
jeux_gagnes			INT
heure_reunion		10:30 2020/12/4	

il n'y a pas de Questions stupides

Q: Pourquoi ne pas utiliser BLOB pour toutes mes valeurs de texte ?

R: C'est un gâchis d'espace. Un VARCHAR ou un CHAR occupe une quantité d'espace précise, pas plus de 256 caractères. Par contre, un BLOB occupe beaucoup plus d'espace de mémoire. Avec le développement de votre base de données, vous courez le risque de finir par manquer d'espace sur votre disque dur. De plus vous ne pouvez pas lancer certaines opérations de chaînes sur les BLOB qui sont possibles avec les VARCHAR et les CHAR (vous en apprendrez davantage plus tard).

Q: Pourquoi ai-je besoin de types numériques comme INT et DEC ?

R: On en revient à l'efficacité et au stockage de la base de donées. Choisir le type de données le plus adapté pour chaque colonne de votre table va diminuer la taille de la table et accélerer les opérations sur vos données.

Q: Est-ce que c'est tout ? Ce sont tous les types ?

R: Non mais ce sont les plus importants. Les types de données diffèrent aussi légèrement selon les SGBDR, vous devrez donc consulter votre documention pour plus d'informations. Nous recommandons *SQL en concentré* (Digit Books), un livre de référence particulièrement bon qui précise les différences entre les SGBDR.

QUEL TYPE DE DONNÉES ?

Un texte peut ne pas toujours avoir 50 caractères de longueur, donc nous utilisons VARCHAR pour économiser de l'espace dans la base de données. Vous pourriez aussi avoir utilisé CHAR ici et avoir choisi une longueur précise.

Déterminez quel est le type de données le plus logique pour chaque colonne. Pendant que vous y êtes, complétez les autres infos manquantes.

Nom de colonne	Description	Exemple	Meilleur choix de type de données
prix	Le prix d'un produit à vendre	5678.39	DEC(6,2)
code_postal	Cinq caractères	33550	VARCHAR(5)
poids_atomique	Poids atomique d'un élément avec jusqu'à 6 décimales	4.002602	DEC(10, 6)
commentaires	Gros bloc de texte de plus de 255 caractères	Auguste, je suis à la réunion des actionnaires. Ils viennent de faire une démo et il y avait des petits canards en plastiques qui volaient à l'écran. C'est ce que tu appelles une blague ? Tu devrais aller faire un tour sur Monster.com.	BLOB
quantite	Nombre d'exemplaires de ce produit en stock	239	INT
tva	Un pourcentage	19.6	DEC(4, 2)
titre_livre	Une chaîne de caractères	SQL Tête la première	VARCHAR(50)
sexe	Un caractère, soit H soit F	H	CHAR(1)
numero_tel	Dix chiffres sans ponctuation	0505552367	CHAR(10)
initiales	Abréviation de deux caractères pour des initiales	TX, CA	CHAR(2)
anniversaire	Mois, jour, année	11/22/2006	DATE
jeux_gagnes	Un entier représentant le nombre de jeux gagnés	15	INT
heure_reunion	Une heure et un jour	10:30 2020/12/4	DATETIME

Un numéro de téléphone aura toujours exactement cette longueur. Nous le considérons comme une chaîne de caractères ~~parce que nous~~ n'avons pas besoin de faire des opérations mathématiques dessus, ~~même si c'est un nombre~~.

TIMESTAMP est le plus souvent utilisé pour saisir l'heure actuelle. DATETIME correspond mieux pour enregistrer un événement futur.

POINTS D'IMPACT

- Classez vos données par catégories avant de créer votre table. Accordez une attention particulière au type de données de chaque colonne.
- Utilisez l'instruction **CREATE DATABASE** pour créer la base de données qui va contenir toutes vos tables.
- Utilisez l'instruction **USE DATABASE** pour entrer dans votre base de données et créer votre table.
- Toutes les tables sont créées avec une instruction **CREATE TABLE**, contenant les noms de colonnes et leur type de données correspondant.
- Parmi les types de données les plus courants, on trouve **CHAR**, **VARCHAR**, **BLOB**, **INT**, **DEC**, **DATE** et **DATETIME**. Chacun d'entre eux a des règles spécifiques pour ce qui peut y rentrer.

Attendez un instant. Où est la table que je viens de créer dans la base de données liste_greg ? Je veux vérifier que j'ai bien tout mis à la bonne place là dedans.

Bien vu. Vérifier votre travail est important.

Pour voir à quoi ressemble la table `mes_contacts` que vous avez créée, vous pouvez utiliser la commande **DESC** :

```
DESC mes_contacts;
```

DESC est l'abréviation de DESCRIBE (décrire)

Essayez.

```
Fichier Edition Fenêtre Aide DescRiption
> DESC mes_contacts;
```

DESCription de votre table

Une fois que vous avez entré la commande DESC, vous voyez quelque chose qui ressemble à ceci :

Ne vous préoccupez pas de cela pour l'instant ; nous allons y venir sous peu.

Fichier Edition Fenêtre Aide DescRiption

```
> DESC mes_contacts;
+-----------------+-------------+------+-----+---------+-------+
| Column          | Type        | Null | Key | Default | Extra |
+-----------------+-------------+------+-----+---------+-------+
| nom             | varchar(30) | YES  |     | NULL    |       |
| prenom          | varchar(20) | YES  |     | NULL    |       |
| email           | varchar(50) | YES  |     | NULL    |       |
| anniversaire    | date        | YES  |     | NULL    |       |
| profession      | varchar(50) | YES  |     | NULL    |       |
| lieu            | varchar(50) | YES  |     | NULL    |       |
| statut          | varchar(20) | YES  |     | NULL    |       |
| centres_interet | varchar(100) | YES |     | NULL    |       |
| cherche         | varchar(100) | YES |     | NULL    |       |
+-----------------+-------------+------+-----+---------+-------+
9 rows in set (0.07 sec)
```

MUSCLEZ VOS NEURONES

Qu'en pensez-vous ? Quels genres de problèmes pourraient découler de l'ajout d'une nouvelle colonne ?

SQL sur le frigo

Le code pour créer la base de données et la table avec la nouvelle colonne sexe se trouve sur des aimants, mélangés sur le frigo. Pouvez-vous reconstruire les fragments de code pour le faire fonctionner ? Certains des points-virgules et des parenthèses sont tombés par terre et ils sont tellement petits qu'on ne les a pas retrouvés, alors n'hésitez pas à en ajouter autant qu'il vous en faut !

```
email VARCHAR(50)
anniversaire DATE
USE liste_greg
prenom VARCHAR(20)
nom VARCHAR(30)
centres_interet VARCHAR(100)
cherche VARCHAR(100)
statut VARCHAR(20)
CREATE DATABASE liste_greg
profession VARCHAR(50)
lieu VARCHAR(50)
CREATE TABLE mes_contacts
sexe CHAR(1)
```

Si vous avez fini, essayez de taper le nouveau code CREATE TABLE dans votre console SQL pour ajouter la nouvelle colonne sexe !

SQL sur le frigo - Solution

Votre mission était de réassembler les fragments de code pour fabriquer le code qui pourrait créer la base de données et la table avec la nouvelle colonne sexe.

liste_greg existe déjà.

Voici le code réassemblé. Vérifiez votre réponse, puis continuez à lire...

```
CREATE DATABASE liste_greg

USE liste_greg

CREATE TABLE mes_contacts
(
    nom VARCHAR(20),
    prenom VARCHAR(30),
    email VARCHAR(50),
    anniversaire DATE,
    sexe CHAR(1),
    profession VARCHAR(50),
    lieu VARCHAR(50),
    statut VARCHAR(20),
    centres_interet VARCHAR(100),
    cherche VARCHAR(100)
);
```

Vous ne pouvez pas recréer une table ou une base de données existante !

Avez-vous essayé d'entrer la nouvelle instruction CREATE TABLE ? Si oui, vous savez déjà que la solution de l'exercice ne va pas vous aider à ajouter la nouvelle colonne.

Si vous l'avez entrée dans votre console, vous avez probablement vu quelque chose comme ça :

```
Fichier  Edition  Fenêtre  Aide OhMerde !
> CREATE TABLE mes_contacts
    -> (
    ->     nom VARCHAR(30),
    ->     prenom VARCHAR(20),
    ->     email VARCHAR(50),
    ->     sexe CHAR(1),
    ->     anniversaire DATE,
    ->     profession VARCHAR(50),
    ->     lieu VARCHAR(50),
    ->     statut VARCHAR(20),
    ->     centres_interet VARCHAR(100),
    ->     cherche VARCHAR(100)
    -> );
ERROR 1050 (42S01): Table 'mes_contacts' already exists
```

La nouvelle colonne pour le sexe.

Oh oh. Cette instruction vous donne un message d'erreur. On dirait que la table n'a pas été créée.

il n'y a pas de
Questions stupides

Q: À propos de cet exercice SQL sur le frigo, pourquoi ai-je obtenu une erreur ?

R: Vous ne pouvez pas créer une table qui existe déjà et une fois que vous avez créé une base de données, vous n'avez pas besoin de la créer une deuxième fois. L'oubli du point-virgule est une autre erreur possible. De même, vérifiez si vous avez fait une faute de frappe sur un des mots-clés de SQL.

Q: Pourquoi n'y a-t-il pas de virgule après « cherche VARCHAR(100) » comme après toutes les autres colonnes ?

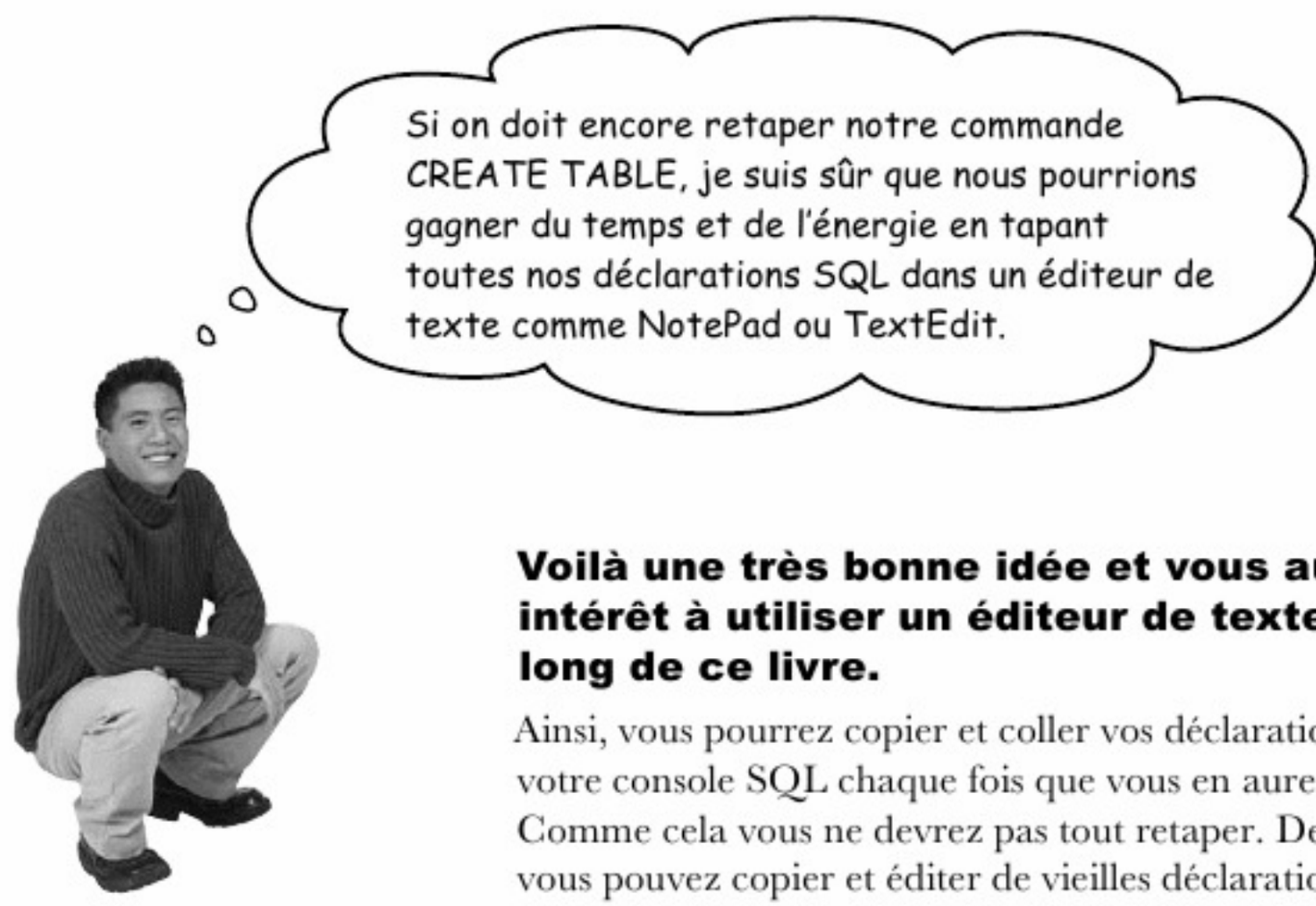
R: La colonne « cherche » est la dernière colonne avant la parenthèse fermante. Celle-ci dit au SGBDR que c'est la fin de l'instruction, on n'a donc pas besoin de virgule.

Q: Alors, y a-t-il un moyen d'ajouter la colonne oubliée ou est-ce que je vais devoir tout recommencer ?

R: Vous allez devoir tout recommencer. Mais avant que vous puissiez créer la table avec la colonne sexe supplémentaire, vous devez vous débarrasser de l'ancienne. Comme il n'y a pas encore de données dans la table, nous pouvons simplement supprimer la vieille table et recommencer.

Q: Mais si j'ai une table contenant des données et que j'ai besoin d'ajouter une colonne ? Y a-t-il un moyen de le faire sans supprimer toute la table et recommencer ?

R: Excellente question ! Il existe un moyen de modifier votre table sans altérer ses données. Nous allons y venir un peu plus tard mais pour l'instant, puisque notre table est vide, nous allons nous en débarrasser et en créer une nouvelle.



Voilà une très bonne idée et vous aurez intérêt à utiliser un éditeur de texte tout au long de ce livre.

Ainsi, vous pourrez copier et coller vos déclarations dans votre console SQL chaque fois que vous en aurez besoin. Comme cela vous ne devrez pas tout retaper. De même, vous pouvez copier et éditer de vieilles déclarations SQL pour en faire de nouvelles.

Dites au revoir à la vieille table et bonjour à la nouvelle

1 Se débarrasser d'une table est beaucoup plus facile que d'en créer une. Utilisez cette simple commande :

La commande pour supprimer votre table...

...et le nom de la table à supprimer.

```
DROP TABLE mes_contacts;
```

N'oubliez pas le point-virgule.

```
Fichier Edition Fenêtre Aide AuRevoirLaTable
> DROP TABLE mes_contacts;
Query OK, 0 rows affected (0.12 sec)
```

`DROP TABLE` fonctionne qu'il y ait des données dans votre table ou pas, donc *utilisez cette commande avec une extrême prudence*. Une fois que la table est supprimée, elle disparaît, ainsi que toutes les données qu'elle contenait.

DROP TABLE supprime votre table et toutes ses données !

2 Maintenant vous pouvez entrer votre nouvelle instruction **`CREATE TABLE`** :

```
Fichier Edition Fenêtre Aide Succès
> CREATE TABLE mes_contacts
    -> (
    ->    nom VARCHAR(30),
    ->    prenom VARCHAR(20),
    ->    email VARCHAR(50),
    ->    sexe CHAR(1),
    ->    anniversaire DATE,
    ->    profession VARCHAR(50),
    ->    lieu VARCHAR(50),
    ->    statut VARCHAR(20),
    ->    centres_interet VARCHAR(100),
    ->    cherche VARCHAR(100)
    -> );
Query OK, 0 rows affected (0.05 sec)
```

Cela a marché cette fois.

Qui suis-je?

Un groupe de mots-clés et de types de données SQL, entièrement déguisés, s'amusent à jouer à « Qui suis-je ? » Ils vous donnent un indice et vous essayez de deviner qui ils sont à partir de ce qu'ils disent. Supposez qu'ils vous disent toujours la vérité. S'ils disent quelque chose qui peut être vrai pour plusieurs d'entre eux, notez tous ceux pour qui la phrase est vraie. Complétez les espaces en face des phrases avec les noms d'un ou de plusieurs participants.

Les participants de ce soir :

CREATE DATABASE, USE DATABASE, CREATE TABLE, DESC, DROP TABLE, CHAR, VARCHAR, BLOB, DATE, DATETIME, DEC, INT

	Nom
J'ai votre numéro.	.....
Je peux me débarrasser de vos tables indésirables.	.....
Les questions V ou F sont mes préférées.	.....
Je me souviens de l'anniversaire de votre mère.	.....
J'ai toute la table entre mes mains.	.....
Les nombres sont cools mais je déteste les fractions.	.....
J'aime les longues explications pleines de mots.	.....
C'est l'endroit où l'on peut tout conserver.	.....
La table n'existerait pas sans moi.	.....
Je connais l'heure exacte de votre rendez-vous chez le dentiste la semaine prochaine.	.....
Les comptables m'aiment bien.	.....
Je vous permets de jeter un coup d'œil au format de votre table.	.....
Sans nous, vous ne pourriez même pas créer une table.	.....

→ Réponses page 51.

Anatomie d'une instruction

Pour ajouter des données dans votre table, vous utilisez l'instruction INSERT

Elle fait plus ou moins ce que son nom indique : insérer. Observez l'instruction ci-dessous pour voir comment fonctionne chaque partie. Les valeurs du deuxième ensemble de parenthèses doivent être dans le ***même ordre que les noms des colonnes.***

La commande ci-dessous n'est pas une commande réelle, c'est un modèle d'instruction pour vous montrer le format d'une instruction `INSERT`.

Les mots-clés INSERT INTO commencent l'instruction.

Le nom de votre table. Dans le cas de Greg, ce sera mes_contacts.

Cette partie suivante est la liste des noms de vos colonnes séparés par des virgules. Vous savez déjà que la liste de Greg aura des colonnes comme prenom, nom et email.

D'autres noms de colonnes suivent, pas de virgules après le dernier.

```
INSERT INTO votre_table (nom_colonne1, nom_colonne2,...)
       VALUES ('valeur1', 'valeur2',...);
```

Un autre mot-clé. Il signale que les valeurs des colonnes suivent.

Cette partie suivante est la liste de vos valeurs séparées par des virgules. Dans le cas de Greg, la liste contiendra les informations de ses pense-bêtes.

Les guillemets simples sont corrects. Utilisez-les à chaque fois que vous insérez du texte, même si c'est un caractère unique comme 'H' ou 'F'.

D'autres valeurs suivent, pas de virgule après la dernière.

Le point-virgule habituel terminant l'instruction.

IMPORTANT : les valeurs doivent être dans le même ordre que les noms de colonnes.

qui fait quoi ?

Avant de pouvoir écrire votre instruction INSERT, vous devez faire correspondre vos noms de colonnes et vos valeurs.

Colonnes	Valeurs
prenom	'relations, amis'
statut	'Bouquet'
cherche	'1980-09-05'
sexe	'Écrivain technique'
anniversaire	'Carole'
nom	'Célibataire'
lieu	'F'
centres_interet	'Nice, 06'
profession	'carole_bouquet@pizzaprestissima.com'
email	'kayak, reptiles'

QUI FAIT QUOI ?

Avant de pouvoir écrire votre instruction INSERT, vous devez faire correspondre vos noms de colonnes et vos valeurs.

Colonnes	Valeurs
prenom	'relations, amis'
statut	'Bouquet'
cherche	'1980-09-05'
sexe	'Écrivain technique'
anniversaire	'Carole'
nom	'Célibataire'
lieu	'F'
centres_interet	'Nice, 06'
profession	'carole_bouquet@pizzaprestissima.com'
email	'kayak, reptiles'

Le type DATE exige un format spécifique. Consultez votre documentation SQL pour les particularités.

N'oubliez pas. Vous avez besoin des guillemets simples pour les valeurs à caractère unique.

Créer l'instruction INSERT

Vos noms de colonnes se trouvent entre les premières parenthèses, séparés par des virgules.

Vous pouvez appuyer sur entrée avant la parenthèse ouvrante pour rendre le code plus facile à lire dans la fenêtre de votre console.

```
INSERT INTO mes_contacts
(nom, prenom, email, sexe, anniversaire,
profession, lieu, statut, centres_interet,
cherche)
VALUES
('Bouquet', 'Carole', 'carole_bouquet@
pizzaprestissima.com', 'F', '1980-09-05',
'Écrivain technique', 'Nice, 06', 'Célibataire',
'kayak, reptiles', 'relations, amis');
```

Pressez entrée après la parenthèse fermante des colonnes et encore une fois après VALUES pour rendre le code plus facile à lire.

Les valeurs pour chaque colonne se trouvent dans le deuxième ensemble de parenthèses et sont également séparées par des virgules.

Toute valeur qui va dans une colonne VARCHAR, CHAR, DATE ou BLOB est entourée par des guillemets simples.

Ouvrez les yeux !

L'ordre est important !

*Les **valeurs** doivent être énumérées **exactement** dans le **même ordre** que les **noms de colonnes**.*

À faire à la maison

Voici un moyen d'ajouter une ligne à votre table. Essayez de taper l'instruction vous-même. Tapez-la d'abord dans un éditeur de texte pour ne pas avoir à tout retaper si vous faites une erreur. Portez une attention particulière aux guillemets simples et aux virgules. Notez ici la réponse que vous obtenez :

Vous venez de me dire que les valeurs **CHAR**, **VARCHAR**, **DATE** et **BLOB** sont entourées par des **guillemets simples** dans l'instruction **INSERT**. Alors cela signifie que les **valeurs numériques** comme **DEC** et **INT n'utilisent pas de guillemets** ?

Exactement.

Voici une instruction `INSERT` que vous pourriez utiliser si vous aviez une table d'achats de crêpes. Notez que, parmi les valeurs, les nombres qui correspondent aux colonnes représentant le nombre de crêpes achetées et le prix n'ont pas de guillemets.

La colonne nombre est un INT, puisque l'on n'achète normalement pas de morceaux de crêpes. Nous n'avons donc pas besoin de décimales.

La colonne prix est DEC(4,2) ce qui signifie qu'elle est large de quatre chiffres, dont deux décimales.

```
INSERT INTO achats_crepes
(type_crepe, nombre, garniture, prix)
VALUES
('fourrée', 3, 'confiture', 3.50);
```

Les valeurs insérées dans les colonnes nombre et prix n'ont pas besoin de guillemets !

À vos crayons !

Votre SGBDR SQL vous dit quand quelque chose ne va pas dans votre instruction, mais il est parfois un peu vague. Regardez chacune des déclarations INSERT ci-dessous. Essayez d'abord de deviner ce qui ne va pas dans l'instruction puis essayez de l'entrer pour voir ce que dit votre SGBDR.

```
INSERT INTO mes_contacts

(nom, prenom, email, sexe, anniversaire, profession, lieu, statut, centres_interet,
 cherche) VALUES ('Bouquet', 'Carole', 'carole_bouquet@pizzaprestissima.com', 'F',
'1980-09-05', 'Écrivain technique', 'Célibataire', 'kayak, reptiles', 'relations,
 amis');
```

Qu'est-ce qui ne va pas ? .....

Votre SGBDR dit : .....

```
INSERT INTO mes_contacts

(nom, prenom, sexe, anniversaire, profession, lieu, statut, centres_interet, cherche)
 VALUES ('Bouquet', 'Carole', 'carole_bouquet@pizzaprestissima.com', 'F', '1980-09-05',
'Écrivain technique', 'Nice, 06', 'Célibataire', 'kayak, reptiles', 'relations, amis');
```

Qu'est-ce qui ne va pas ? .....

Votre SGBDR dit : .....

```
INSERT INTO mes_contacts

(nom, prenom, email, sexe, anniversaire, profession, lieu, statut, centres_interet,
 cherche) VALUES ('Bouquet', 'Carole', 'carole_bouquet@pizzaprestissima.com', 'F',
'1980-09-05', 'Écrivain technique' 'Nice, 06', 'Célibataire', 'kayak, reptiles',
'relations, amis');
```

Qu'est-ce qui ne va pas ? .....

Votre SGBDR dit : .....

```
INSERT INTO mes_contacts

(nom, prenom, email, sexe, anniversaire, profession, lieu, statut, centres_interet,
 cherche) VALUES ('Bouquet', 'Carole', 'carole_bouquet@pizzaprestissima.com', 'F',
'1980-09-05', 'Écrivain technique', 'Nice, 06', 'Célibataire', 'kayak, reptiles',
'relations, amis);
```

Qu'est-ce qui ne va pas ? .....

Votre SGBDR dit : .....

Si celle-ci fait ramer votre SGBDR, essayez de taper un guillemet simple suivi par un point-virgule après le reste de l'instruction.

À vos crayons ! Solution

Votre SGBDR SQL vous dit quand quelque chose ne va pas dans votre instruction mais il est parfois un peu vague. Regardez chacune des déclarations INSERT ci-dessous. Essayez d'abord de deviner ce qui ne va pas dans l'instruction puis essayez de l'entrer pour voir ce que dit votre SGBDR.

```
INSERT INTO mes_contacts

(nom, prenom, email, sexe, anniversaire, profession, lieu, statut, centres_interet,
 cherche) VALUES ('Bouquet', 'Carole', 'carole_bouquet@pizzaprestissima.com', 'F',
'1980-09-05', 'Écrivain technique', 'Célibataire', 'kayak, reptiles', 'relations,
 amis');
```

Nous avons une colonne lieu dans la liste des colonnes mais pas de lieu dans la liste des valeurs. Il nous manque une valeur.

Qu'est-ce qui ne va pas ? Il manque une valeur lieu

Votre SGBDR dit : ERROR 1136 (21S01): Column count doesn't match value count at row 1

Remarquez que beaucoup de problèmes différents donnent la même erreur. Faites attention à la typo ; les erreurs peuvent être difficiles à identifier.

```
INSERT INTO mes_contacts

(nom, prenom, sexe, anniversaire, profession, lieu, statut, centres_interet, cherche)
 VALUES ('Bouquet', 'Carole', 'carole_bouquet@pizzaprestissima.com', 'F', '1980-09-05',
'Écrivain technique', 'Nice, 06', 'Célibataire', 'kayak, reptiles', 'relations, amis');
```

Cette fois nous avons une valeur pour toutes les colonnes mais il nous manque la colonne email dans la liste des colonnes.

Qu'est-ce qui ne va pas ? Il manque email dans la liste des colonnes

Votre SGBDR dit : ERROR 1136 (21S01): Column count doesn't match value count at row 1

```
INSERT INTO mes_contacts

(nom, prenom, email, sexe, anniversaire, profession, lieu, statut, centres_interet,
 cherche) VALUES ('Bouquet', 'Carole', 'carole_bouquet@pizzaprestissima.com', 'F',
'1980-09-05', 'Écrivain technique' 'Nice, 06', 'Célibataire', 'kayak, reptiles',
'relations, amis');
```

Pas de virgule dans la liste des valeurs entre 'Écrivain technique' et 'Nice, 06'

Qu'est-ce qui ne va pas ? Il manque une virgule entre deux valeurs

Votre SGBDR dit : ERROR 1136 (21S01): Column count doesn't match value count at row 1

```
INSERT INTO mes_contacts

(nom, prenom, email, sexe, anniversaire, profession, lieu, statut, centres_interet,
 cherche) VALUES ('Bouquet', 'Carole', 'carole_bouquet@pizzaprestissima.com', 'F',
'1980-09-05', 'Écrivain technique', 'Nice, 06', 'Célibataire', 'kayak, reptiles',
'relations, amis);
```

Qu'est-ce qui ne va pas ? Il manque un guillement simple après la dernière valeur

Votre SGBDR dit : ERROR 1064 (42000): You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near " at line 4

Variations sur une instruction INSERT

Il existe trois variantes de l'instruction **INSERT** que vous devriez connaître.

❶ Changer l'ordre des colonnes

Vous pouvez changer l'ordre des noms de colonnes, tant que les valeurs correspondantes à chaque colonne suivent cet ordre !

```
INSERT INTO mes_contacts

(centres_interet, prenom, nom, sexe, email, anniversaire,
profession, lieu, statut, cherche)

VALUES

('kayak, reptiles', 'Carole', 'Bouquet', 'F',
'carole_bouquet@pizzaprestissima.com', '1980-09-05', 'Écrivain
technique', 'Nice, 06', 'Célibataire', 'relations, amis');
```

Vous voyez l'ordre des noms de colonnes ? Maintenant regardez les valeurs : elles sont dans le même ordre. Donc tant que les valeurs correspondent aux noms de colonnes, l'ordre dans lequel vous les insérez n'a pas d'importance, ni pour vous ni pour votre SGBDR SQL !

❷ Omettre les noms de colonnes

Vous pouvez omettre la liste des noms de colonnes mais les valeurs doivent **toutes** être représentées et ceci **dans le même ordre** que **quand vous avez ajouté les colonnes**. (Vérifiez l'ordre page 37 si vous hésitez.)

```
INSERT INTO mes_contacts

VALUES

('Bouquet', 'Carole', 'carole_bouquet@pizzaprestissima.
com', 'F', '1980-09-05', 'Écrivain technique', 'Nice, 06',
'Célibataire', 'kayak, reptiles', 'relations, amis');
```

Nous avons omis tous les noms de colonnes mais si vous faites cela, vous devez inclure TOUTES les valeurs et ceci dans l'ORDRE EXACT ou ils se trouvent ~~dans la table~~ !

❸ Omettre des colonnes

Vous pouvez insérer quelques colonnes et en exclure d'autres.

```
INSERT INTO mes_contacts

(nom, prenom, email)

VALUES

('Bouquet', 'Carole', 'carole_bouquet@
pizzaprestissima.com');
```

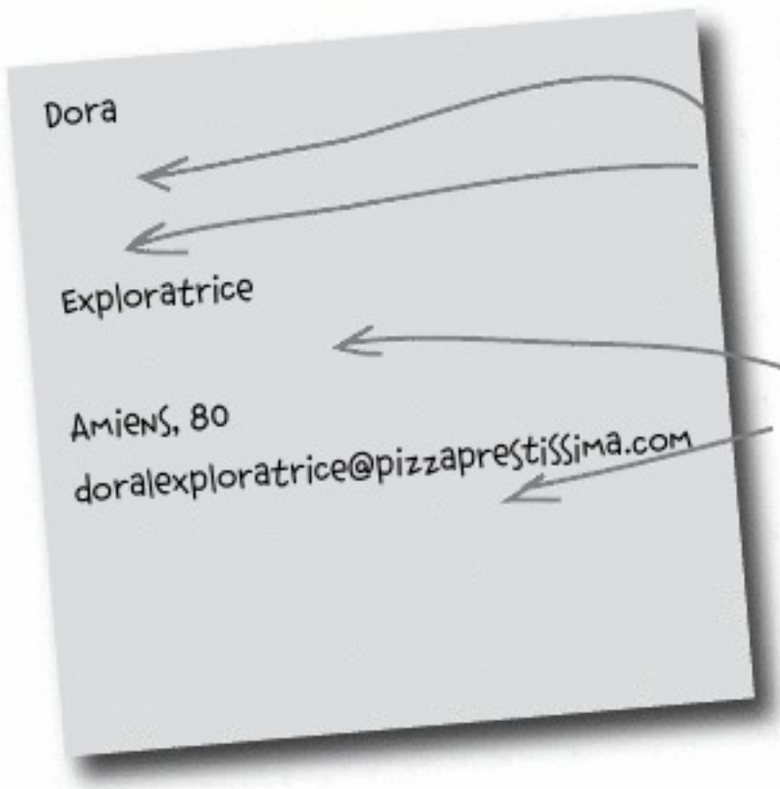
Cette fois, nous insérons seulement une partie de nos données. Comme votre SGBDR SQL ne sait pas quelle partie, vous devez le lui dire en spécifiant les noms de colonnes et les valeurs que vous entrez.

MUSCLEZ VOS NEURONES

À votre avis, que contient la table dans les colonnes auxquelles vous n'assignez pas de valeur ?

Colonnes sans valeurs

Saisissons un enregistrement dans la table `mes_contacts` à partir de ce pense-bête incomplet :



Il manque le nom, l'anniversaire et nous ne pouvons pas déterminer le sexe avec certitude non plus.

Il manque également statut, centres_interet et cherche colonnes.

COLONNES :	VALEURS :
nom	?
prenom	'Dora'
email	'doralexploratrice@pizzaprestissima.com'
sexe	?
anniversaire	?
profession	'Exploratrice'
lieu	'Amiens, 80'
statut	?
centres_interet	?
cherche	?

Comme il manque des données sur le pense-bête, Greg va devoir entrer un enregistrement incomplet. Mais ce n'est pas grave, il pourra ajouter les informations manquantes plus tard.

Nous utilisons la version d'INSERT où nous n'avons pas besoin de fournir des données pour toutes les colonnes parce que cela nous permet de n'inclure que les colonnes dont nous connaissons les valeurs.

```
INSERT INTO mes_contacts
(prenom, email, profession, lieu)
VALUES
('Dora', 'doralexploratrice@pizzaprestissima.
com', 'Exploratrice', 'Amiens, 80');
```

```
Fichier Edition Fenêtre Aide PlusDeDonnéesSilVousPlait
> INSERT INTO mes_contacts (prenom, email, profession, lieu)
VALUES ('Dora', 'doralexploratrice@pizzaprestissima.com',
'Exploratrice', 'Amiens, 80');

Query OK, 1 row affected (0.02 sec)
```

Piochez dans votre table avec l'instruction SELECT

Alors, vous voulez voir à quoi ressemble votre table ? Bon, `DESC` ne va plus faire l'affaire parce qu'il ne montre que la *structure* de la table et pas les informations qui s'y trouvent. Vous devriez plutôt utiliser une simple instruction `SELECT` pour voir les données que contient votre table.

Nous voulons sélectionner toutes les données de notre table...

... et l'astérisque demande de TOUT sélectionner.

Le nom de notre table.

```
SELECT * FROM mes_contacts;
```

Ne vous préoccupez pas de ce que fait l'instruction SELECT pour l'instant.

Nous regarderons cela de façon bien plus détaillée dans le chapitre 2. Pour l'instant, détendez-vous et laissez-vous émerveiller par la beauté de votre table quand vous utilisez cette instruction.

Essayez-le vous-même maintenant. Vous devrez allonger votre fenêtre pour voir tous les résultats joliment disposés.

```
Fichier Edition Fenêtre Aide BeaucoupDeNULL
> SELECT * FROM mes_contacts;
+---------+---------+---------+---------+---------+---------+---------+---------+
| nom     | prenom  | email   | sexe    |anniversaire| profession | lieu    | statut  | centres_interet | cherche |
+---------+---------+---------+---------+---------+---------+---------+---------+
| NULL    | Dora    | doralexploratrice@pizzaprestissima.com| NULL | NULL | Exploratrice | Amiens, 80| NULL | NULL |
| Bouquet | Carole  |carole_bouquet@pizzaprestissima.com| F | 1980-09-05 | Ecrivain technique | Nice, 06 | Célibataire | kayak, reptiles | relations, ami |
+---------+---------+---------+---------+---------+---------+---------+---------+
2 rows in set (0.00 sec)
```

Il y a une valeur NULL dans chaque colonne où nous n'avions pas de valeur.

MUSCLEZ VOS NEURONES

Vous savez maintenant que NULL apparaît dans toute colonne sans valeur assignée. À votre avis, que *signifie* réellement NULL ?

Révélations sur SQL

L'interview de cette semaine : Confessions d'un NULL

Tête la première : Bienvenue, NULL. Je dois dire que je ne m'attendais pas à vous voir. Je ne pensais pas que vous existiez vraiment. Le bruit court que vous n'êtes rien de plus qu'un zéro ou qu'un rien du tout.

NULL : Je ne peux pas croire que vous écoutiez de tels mensonges. Oui, je suis là et je suis plutôt réel ! Ainsi vous pensez que je ne suis rien, juste un peu de saleté sous vos pieds ?

TLP : Doucement, calmez-vous. C'est juste que l'on vous voit à chaque fois que quelque chose n'a pas de valeur…

NULL : Bien sûr, il vaut mieux moi que, disons, un zéro ou une chaîne vide.

TLP : Qu'est-ce qu'une chaîne vide ?

NULL : C'est quand vous utilisez deux guillemets simples sans valeur à l'intérieur. C'est toujours une chaîne de caractères, mais de longueur zéro. Comme de définir une valeur pour prenom dans la table mes_contacts à ''.

TLP : Alors vous n'êtes pas un moyen fantaisiste de ne rien dire ?

NULL : Je vous ai dit, je ne suis pas rien ! Je suis quelque chose… Je suis seulement un peu… indéfini, c'est tout.

TLP : Donc vous dites que si je vous comparais à un zéro ou à une chaîne de caractères vide, vous n'y seriez pas égal ?

NULL : Non ! Je ne serais jamais égal à zéro et d'ailleurs, je ne serais jamais non plus égal à un autre NULL. Vous ne pouvez pas comparer un NULL à un autre. Une valeur peut **être** NULL, mais elle n'est jamais **égale à** NULL parce que NULL est une valeur indéfinie ! Vous comprenez ?

TLP : Calmez-vous et laissez-moi éclaircir tout cela. Vous n'êtes pas égal à zéro, vous n'êtes pas une chaîne variable vide et vous n'êtes même pas égal à vous-même ? Cela n'a aucun sens!

NULL : Je sais que c'est déroutant. Voyez-le comme ça : je suis indéfini. Je suis comme l'intérieur d'une boîte jamais ouverte. Il pourrait y avoir n'importe quoi à l'intérieur, donc vous ne pouvez pas comparer une boîte qui n'a pas encore été ouverte à une autre parce que vous ne savez pas ce qu'il y aura à l'intérieur de chacune d'elle. Je pourrais même être vide. Vous ne savez tout simplement pas.

TLP : J'ai entendu dire que parfois vous êtes indésirable. Que peut-être, des fois, vous autres les NULL posez des problèmes.

NULL : Je dois admettre que je me suis trouvé à des endroits où l'on ne voulait pas de moi par le passé. Certaines colonnes devraient toujours avoir des valeurs. Comme nom, par exemple. Il n'y a aucun intérêt à avoir un nom NULL dans une table.

TLP : Alors vous n'allez pas là où l'on ne veut pas de vous ?

NULL : Exact ! Il n'y a qu'à me le dire, mon ami ! Quand vous créez votre table et que vous définissez vos colonnes, passez-moi l'information.

TLP : Vous ne ressemblez pas vraiment à une boîte jamais ouverte.

NULL : Ça suffit maintenant. J'ai des choses à faire, des valeurs à être.

Contrôler votre NULL intérieur

Certaines colonnes de votre table doivent toujours avoir des valeurs. Vous souvenez-vous du pense-bête incomplet pour Dora, sans nom de famille ? Il (ou elle) ne sera pas très facile à retrouver quand vous aurez vingt autres entrées NULL dans les noms de famille de votre table. Vous pouvez facilement configurer votre table pour qu'elle n'accepte pas de valeurs `NULL` pour ses colonnes.

```
CREATE TABLE mes_contacts
(
   nom VARCHAR (30) NOT NULL,
   prenom VARCHAR (20) NOT NULL
);
```

Ajoutez simplement les mots NOT NULL juste après le type de données.

Si vous utilisez cela, vous devez fournir une valeur pour la colonne dans votre instruction INSERT. Sinon vous obtiendrez une erreur.

À vos crayons !

```
CREATE TABLE mes_contacts
(
   nom VARCHAR(30) NOT NULL,
   prenom VARCHAR(20) NOT NULL,
   email VARCHAR(50),
   sexe CHAR(1),
   anniversaire DATE,
   profession VARCHAR(50),
   lieu VARCHAR(50),
   statut VARCHAR(20),
   centres_interet VARCHAR(100),
   cherche VARCHAR(100)
);
```

Regardez chacune des colonnes dans notre commande CREATE TABLE mes_contacts. Lesquelles devraient être configurées en NOT NULL ? Demandez-vous quelles colonnes ne devraient jamais être NULL et entourez-les.

Nous vous en avons donné deux pour commencer, à vous de terminer maintenant. Considérez d'abord les colonnes que vous allez utiliser plus tard pour vos recherches ou celles qui sont uniques.

À vos crayons ! Solution

```
CREATE TABLE mes_contacts
(
    nom VARCHAR(30) NOT NULL,
    prenom VARCHAR(20) NOT NULL,
    email VARCHAR(50),
    sexe CHAR(1),
    anniversaire DATE,
    profession VARCHAR(50),
    lieu VARCHAR(50),
    statut VARCHAR(20),
    centres_interet VARCHAR(100),
    cherche VARCHAR(100)
);
```

Regardez chacune des colonnes dans notre commande CREATE TABLE mes_contacts. Lesquelles devraient être configurées en NOT NULL ? Demandez-vous quelles colonnes ne devraient jamais être NULL et entourez-les.

Nous vous en avons donné deux pour commencer, à vous de terminer maintenant. Considérez d'abord les colonnes que vous allez utiliser plus tard pour vos recherches ou celles qui sont uniques.

Toutes les colonnes doivent être NOT NULL.

Vous allez utiliser TOUTES vos colonnes pour vos recherches. Il est important de s'assurer que vos enregistrements sont complets et que votre table contient de bonnes données...

...mais si vous avez une colonne dont vous savez qu'elle doit être complétée plus tard, vous pouvez vouloir autoriser les valeurs NULL dedans.

NOT NULL apparaît dans DESC

Voilà à quoi la table `mes_contacts` ressemblerait si vous configuriez toutes les colonnes pour qu'elles aient des valeurs `NOT NULL`.

C'est là que nous créons notre table avec NOT NULL dans chaque colonne.

```
Fichier Edition Fenêtre Aide PlusDeNULL
CREATE TABLE mes_contacts
(
   nom VARCHAR(30) NOT NULL,
   prenom VARCHAR(20) NOT NULL,
   email VARCHAR(50) NOT NULL,
   sexe CHAR(1) NOT NULL,
   anniversaire DATE NOT NULL,
   profession VARCHAR(50) NOT NULL,
   lieu VARCHAR(50) NOT NULL,
   statut VARCHAR(20) NOT NULL,
   centres_interet VARCHAR(100) NOT NULL,
   cherche VARCHAR(100) NOT NULL
);
Query OK, 0 rows affected (0.01 sec)

> DESC mes_contacts;
+----------------+--------------+------+-----+---------+-------+
| Column         | Type         | Null | Key | Default | Extra |
+----------------+--------------+------+-----+---------+-------+
| nom            | varchar(30)  | NO   |     |         |       |
| prenom         | varchar(20)  | NO   |     |         |       |
| email          | varchar(50)  | NO   |     |         |       |
| sexe           | char(1)      | NO   |     |         |       |
| anniversaire   | date         | NO   |     |         |       |
| profession     | varchar(50)  | NO   |     |         |       |
| lieu           | varchar(50)  | NO   |     |         |       |
| statut         | varchar(20)  | NO   |     |         |       |
| centres_interet| varchar(100) | NO   |     |         |       |
| cherche        | varchar(100) | NO   |     |         |       |
+----------------+--------------+------+-----+---------+-------+
10 rows in set (0.02 sec)
```

Voici la table décrite. Remarquez le mot NO en-dessous de NULL.

Remplir les blancs avec DEFAULT

Si nous savons qu'une colonne est généralement une valeur particulière, nous pouvons lui assigner une valeur **DEFAULT**. La valeur qui suit le mot-clé DEFAULT est automatiquement insérée dans la table à chaque fois qu'une ligne est ajoutée ***si aucune autre valeur n'est spécifiée***. La valeur par défaut doit être du même type de valeur que la colonne.

```
CREATE TABLE liste_crepes
(
 nom_crepe VARCHAR(15) NOT NULL,
 type_crepe VARCHAR(7) NOT NULL,
 prix_crepe DEC(3,2) NOT NULL DEFAULT 2.00
);
```

Nous voulons nous assurer de toujours avoir une valeur dans cette colonne. Nous pouvons non seulement la rendre NOT NULL, mais aussi lui assigner une valeur DEFAULT de 2€.

Ceci sera la valeur insérée dans la table pour la colonne prix_crepe si aucune autre valeur n'est désignée.

liste_crepes

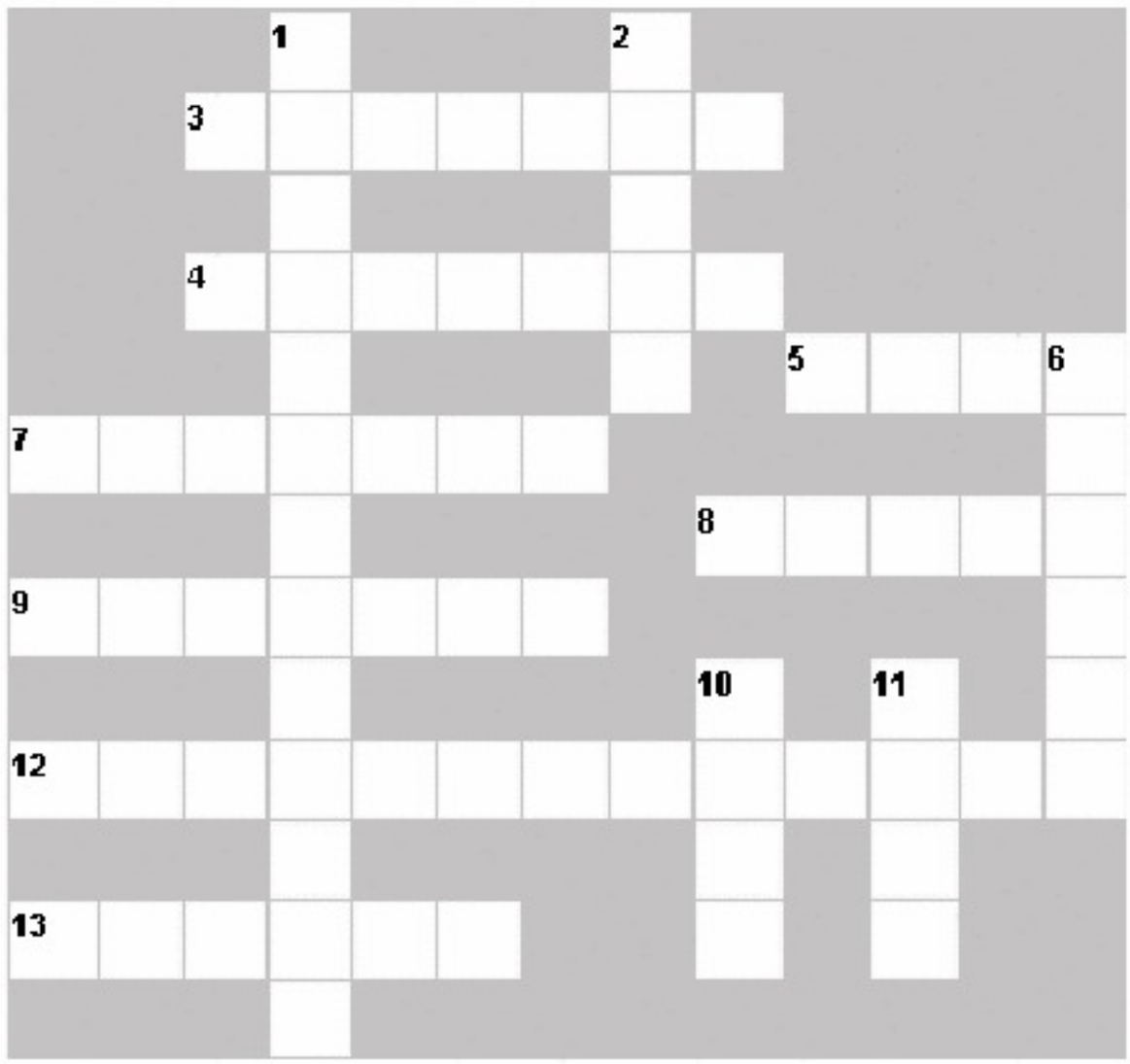
nom_crepe	type_crepe	prix_crepe
Miel amande	fourrée	3.50
Suzette	flambée	2.00
Citron pressé	sirop	2.00
Sirop d'érable	sirop	2.00
Crème de marron	fourrée	4.00

Voici ce à quoi votre table ressemblerait si vous laissiez la valeur de prix_crepe vide en insérant les enregistrements pour les crêpes Suzette, Citron pressé et Sirop d'érable.

Utiliser une valeur DEFAULT complète les colonnes vides avec une valeur spécifique.

SQL-croisés

Prenez le temps de vous asseoir confortablement et activez votre hémisphère gauche. Ce sont vos mots-croisés habituels ; toutes les solutions se trouvent dans ce chapitre.



Définitions horizontales :

3. Peut contenir des données texte jusqu'à 255 caractères de longueur.

4. Utilisez-le dans votre CREATE TABLE pour spécifier une valeur pour votre colonne si aucune autre valeur n'est assignée dans un INSERT.

5. Utilisez ce mot-clé pour voir la table que vous venez de créer.

7. Ce type de données pense que les nombres devraient être entiers mais n'a pas peur des nombres négatifs.

8. Un ensemble unique de colonnes qui décrit les attributs d'une chose unique.

9. Une ... est une donnée conservée par votre table.

12. Terminez toutes vos instructions SQL avec un ...-... .

13. Pour ajouter des données dans votre table, vous utilisez l'instruction ... .

Définitions verticales :

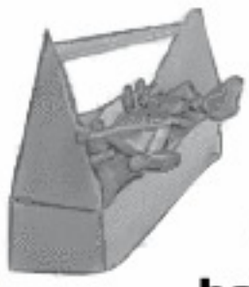
1. Une ... est un récipient qui contient les tables et les autres structures SQL liées à ces tables. (3 mots)

2. C'est la structure, à l'intérieur de votre base de données, qui contient les données, organisées en colonnes et en lignes.

6. Ce mot peut être utilisé devant TABLE et devant DATABASE.

10. Pour vous débarrasser de votre table, utilisez ... TABLE.

11. On ne peut pas comparer un ... à un autre.



Boîte à outils SQL

Le premier chapitre est derrière vous et vous savez déjà comment créer des bases de données et des tables, ainsi que comment y insérer quelques-uns des types de données les plus courants tout en vous assurant que les colonnes qui ont besoin d'une valeur en obtiennent une.

CREATE DATABASE

Utilisez cette instruction pour définir la base de données qui va contenir toutes vos tables.

USE DATABASE

Vous permet d'entrer dans la base de données pour définir toutes vos tables.

CREATE TABLE

Commence à définir votre table mais vous avez aussi besoin de connaître vos NOMS DE COLONNES et TYPES DE DONNÉES. Vous devez les trouver en analysant le genre de données que vous allez entrer dans votre table.

NULL et NOT NULL

Vous avez également besoin de savoir quelles colonnes ne doivent pas accepter de valeurs NULL pour vous aider à trier et fouiller dans vos données. Vous devez définir les colonnes comme NOT NULL quand vous créez votre table.

DEFAULT

Vous permet de préciser une valeur par défaut pour une colonne, utilisée si vous ne fournissez pas de valeur pour la colonne quand vous insérez un enregistrement.

DROP TABLE,

Vous permet de supprimer une table si vous faites une erreur. Mais vous devez faire cela avant de commencer à utiliser des instructions INSERT vous permettant d'ajouter des valeurs dans chaque colonne.

POINTS D'IMPACT

- Si vous voulez voir la structure de votre table, utilisez l'instruction **DESC**.
- L'instruction **DROP TABLE** peut être utilisée pour détruire votre table. Utilisez-la avec prudence !
- Pour insérer vos données dans votre table, utilisez l'une des variantes de l'instruction **INSERT**.
- Une valeur **NULL** est une valeur indéfinie. Elle n'est pas égale à zéro ou à une valeur vide. Une colonne avec une valeur **NULL EST NULL**, mais ***n'est pas* ÉGALE À NULL**.
- Les colonnes auxquelles vous n'assignez pas de valeur dans vos instructions **INSERT** sont définies comme **NULL** par défaut.
- Vous pouvez modifier une colonne pour qu'elle n'accepte pas de valeur **NULL** en utiliant les mots-clés **NOT NULL** quand vous créez votre table.
- Utiliser une valeur **DEFAULT** quand vous créez votre table complète la colonne avec cette valeur si vous insérez un enregistrement sans valeur pour cette colonne.

Qui suis-je ?

Un groupe de mots-clés et de types de données SQL, entièrement déguisés, s'amusent à jouer à « Qui suis-je ? » Ils vous donnent un indice et vous essayez de deviner qui ils sont à partir de ce qu'ils disent. Supposez qu'ils vous disent toujours la vérité. S'ils disent quelque chose qui peut être vrai pour plusieurs d'entre eux, notez tous ceux pour qui la phrase est vraie. Complétez les espaces en face des phrases avec les noms d'un ou de plusieurs participants.

Les participants de ce soir :

CREATE DATABASE, USE DATABASE, CREATE TABLE, DESC, DROP TABLE, CHAR, VARCHAR, BLOB, DATE, DATETIME, DEC, INT

	Nom
J'ai votre numéro.	DEC, INT
Je peux me débarrasser de vos tables indésirables.	DROP TABLE
Les questions V ou F sont mes préférées.	CHAR(1) ← Des points en bonus si vous avez ajouté le (1) !
Je me souviens de l'anniversaire de votre mère.	DATE
J'ai toute la table entre mes mains.	CREATE DATABASE
Les nombres sont cools mais je déteste les fractions.	INT
J'aime les longues explications pleines de mots.	BLOB
C'est l'endroit où l'on peut tout conserver.	CREATE TABLE
La table n'existerait pas sans moi.	CREATE DATABASE
Je connais l'heure exacte de votre rendez-vous chez le dentiste la semaine prochaine.	DATETIME
Les comptables m'aiment bien.	DEC
Je vous permets de jeter un coup d'œil au format de votre table.	DESC
Sans nous, vous ne pourriez même pas créer une table.	CREATE DATABASE, USE DATABASE
	DROP TABLE

Solution des SQL-croisés

			1 B				2 T					
		3 V	A	R	C	H	A	R				
			S				B					
		4 D	E	F	A	U	L	T				
			D				E		5 D	E	S	6 C
7 I	N	T	E	G	E	R						R
			D					8 L	I	G	N	E
9 C	O	L	O	N	N	E						A
			N					10 D		11 N		T
12 P	O	I	N	T	-	V	I	R	G	U	L	E
			É					O		L		
13 I	N	S	E	R	T			P		L		
			S									

2 l'instruction SELECT

Un talent pour la récupération de données

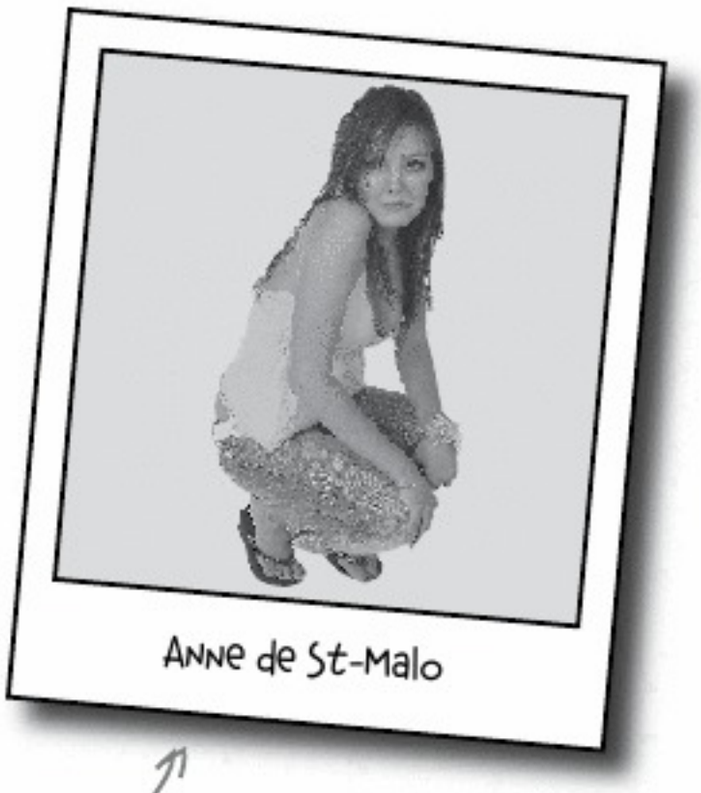
Est-ce vraiment mieux de donner que de recevoir ? S'il s'agit de bases de données, il y a des chances que vous ayez plus souvent besoin de **récupérer vos données** que de les saisir. C'est là que ce chapitre entre en jeu : vous allez rencontrer la puissante instruction **SELECT** et apprendre à **accéder à ces importantes informations** que vous avez entrées dans vos tables. Vous apprendrez même à utiliser **WHERE**, **AND** et **OR** pour accéder sélectivement à vos données et éviter d'afficher celles dont vous n'avez *pas* besoin.

Rendez-vous ou pas rendez-vous?

Greg a terminé d'entrer tous les pense-bêtes dans sa table `mes_contacts`. Il peut maintenant se détendre. Il a deux billets pour un concert et il veut inviter un de ses contacts, une fille de St-Malo.

Il doit trouver son adresse email, donc il utilise l'instruction **SELECT** du premier chapitre pour voir sa table.

```
SELECT * from mes_contacts;
```



Ses coordonées sont dans la table de Greg... quelque part.

Vous ÊTES Greg

Votre mission est d'être Greg. Essayez de trouver Anne de St-Malo dans la première partie de la table mes_contacts, page suivante.

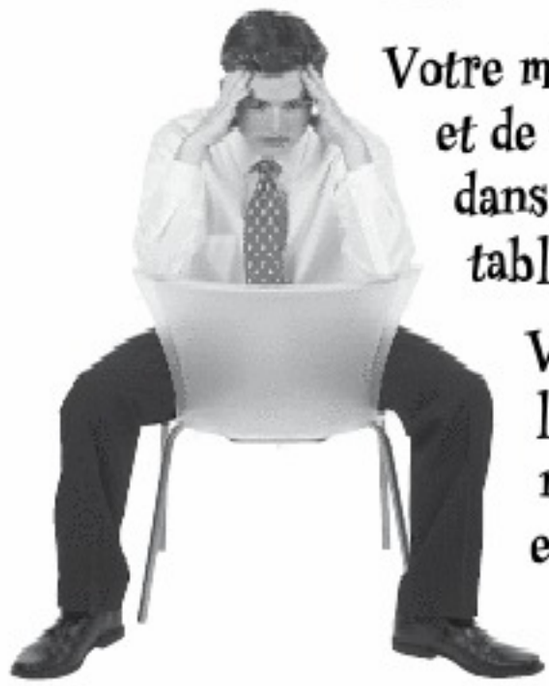
La table mes_contacts a pas mal de colonnes.
Celles-ci ne sont que les premières.

Fichier Edition Fenêtre Aide AnneQui

nom	prenom	email	sexe	lieu
Bouquet	Carole	carole_bouquet@pizzaprestissima.com	F	Nice, 06
Joffe	Pierre	joffe@canardsdesimulation.com	H	Rennes, 35
Nouveau	Amandine	aman2luv@pizzaprestissima.com	F	St-Malo, 35
Garcia	Edouard	ed99@basenhaut.com	H	Nantes, 44
Tourne	Marianne	mamatourne@pizzaprestissima.com	F	St-Malo, 35
Cousteau	Christophe	ccousteau@planchesagogo.com	H	La Rochelle, 1
Toutcoeur	Lucien	hovercraft@pizzaprestissima.com	H	Rennes, 35
Toge	Anne	Anne_Toge@limosbrillantes.com	F	St-Malo, 35
Dupont	André	andredupont@objectville.net	H	Paris, 75
Potumbo	Thomas	undmespot@distribon.net	H	Orléans, 45
Leblanc	Hélène	pirateencolere@pizzaprestissima.com	F	St-Malo, 35
Lamotte	Christian	christian@cafestarbuzz.com	H	Paris, 75
Renard	Anna	anna_renard@fabriqueduchocolat.com	F	St-Malo, 35
Fort	Bertrand	bf@vas-y-cheri.com	H	Le Havre, 76
Lavilliers	Anne	al86@objectville.net	H	Strasbourg, 67
DeBoner	Déborah	debmonstre@pizzaprestissima.com	F	Lille, 59
Tedesco	Jeanne	jeannetedesco@cafestarbuzz.com	F	Saint-Tropez, 8
Talwar	Vikram	vikt@cafestarbuzz.com	H	Nice, 06
Szwed	Jean	szwed_jean@objectville.net	H	Paris, 75
Charles	Diane	charlesdi@distribon.net	F	Lyon, 69
Neige	Edward	hommedesneiges@bardescaraibes.com	H	Biarritz, 64
Otto	Grégory	greg0098@objectville.net	H	Dunkerque, 59
Hardy	Anne	anneh@basenhaut.com	F	St-Malo, 35
Afere	Marie	pasuneafere@cafestarbuzz.com	F	Boulogne, 62
Jagel	Anna	filledereve@pizzaprestissima.com	F	St-Malo, 35
Melfi	Julien	drmelfi@basenhaut.com	H	Clermont-Ferra
Olivier	Louis	louis_olivier@stationmeteo.com	H	Saint-Nectaire
Dubois	Anne	anned@cafestarbuzz.com	F	St-Malo, 35
Ricci	Jean-Pierre	ricci@bardescaraibes.com	H	Dijon, 21
Reno	Rose	rose23@objectville.net	F	Nice, 06
Moss	Zelda	zelda@stationmeteo.com	F	Grenoble, 38
Lejour	Désiré	desirelanuit@pizzaprestissima.com	H	Grasse, 06
Bolger	Eugénie	eugenie@fabriqueduchocolat.com	F	La Rochelle, 1
Sardou	Anne	annesardou@pizzaprestissima.com	F	St-Malo, 35
Booling	Linda	linda@bardescaraibes.com	F	Evian, 74
Gares	Fred	fgares@objectville.net	H	Rennes, 35
Jacob	Anne	anne99@objectville.net	F	Rennes, 35
[illegible]	[illegible]	[illegible]	H	Cannes, 06

Ceci n'est pas la fin de la table ! Greg avait BEAUCOUP de pense-bêtes.

Vous ÊTES Greg - Solutions



Votre mission était d'être Greg et de trouver Anne de St-Malo dans la première partie de la table mes_contacts.

Vous deviez trouver toutes les Anne de St-Malo et noter leur noms, prénoms et adresses email.

Toge, Anne : Anne_Toge@limosbrillantes.com

Hardy, Anne : anneh@basenhaut.com

Dubois, Anne : anned@cafestarbuzz.com

Sardou, Anne : annesardou@pizzaprestissima.com

Voici toutes les Anne et leurs adresses email.

Greg cherche Anne avec un "e". Si vous avez trouvé des Anna, vous deviez les ignorer.

Prendre contact

Cela a pris **bien trop de temps** et c'était **extrêmement fastidieux**. Il est aussi très possible que Greg ait raté certaines des Anne qui correspondent, y compris celle qu'il cherche.

Maintenant que Greg a toutes leurs adresses email, il envoie un email aux Anne et découvre…

À : Toge, Anne <Anne_Toge@limosbrillantes.com>
De : Greg <greg@listegreg.com>
Sujet : Est-ce qu'on s'est rencontrés au Starbuzz ?

Je sors en ce moment avec un gars formidable qui s'appelle Jimmy Bûcheron. Nous nous sommes rencontrés à une fête à l'université. En fait, il était à [illegible] et je passais simplement… tu sais comment

À : Sardou, Anne <annesardou@pizzaprestissima.com>
De : Greg <greg@listegreg.com>
Sujet : Est-ce qu'on s'est rencontrés au Starbuzz ?

Justement je cherchais un petit taquin comme toi ! Passe me prendre à cinq heures et on se préparera un truc vite fait.

À : Hardy, Anne <anneh@basenhaut.com>
De : Greg <greg@listegreg.com>
Sujet : Est-ce qu'on s'est rencontrés au Starbuzz ?

Je ne suis pas la Anne que tu cherches, mais je suis sûre qu'elle est adorable. Si ça ne fonctionne pas entre vous, contacte-moi

À : Dubois, Anne <anned@cafestarbuzz.com>
De : Greg <greg@listegreg.com>
Sujet : Est-ce qu'on s'est rencontrés au Starbuzz ?

Bien sûr que je me souviens de toi ! Si seulement tu m'avais contactée plus tôt. J'ai des projets avec mon ex-petit-ami qui veut qu'on se remette ensemble.

Musclez vos neurones

Pouvez-vous trouver un moyen d'écrire une requête SQL pour n'afficher que les enregistrements qui ont le prénom « Anne » ?

Un peu plus SELECT

Voici une instruction `SELECT` qui aurait aidé Greg à trouver Anne bien plus vite qu'en lisant minutieusement cette immense table toute entière à la recherche des Anne. Dans l'instruction, nous utilisons une ***clause* WHERE *qui donne quelque chose de spécifique à rechercher au SGBDR***. Pour nous, cela réduit les résultats et **ne renvoie que les lignes qui correspondent à la condition.**

Le signe égal dans la clause `WHERE` est utilisé pour tester si chaque valeur de la colonne `prenom` est égale ou correspond au texte `'Anne'`. Si oui, l'ensemble de la ligne est renvoyée. Sinon, la ligne n'est pas renvoyée.

Voici le nom de la table.

```
SELECT * FROM mes_contacts
WHERE prenom = 'Anne';
```

Ajoutez le point-virgule et appuyez sur Entrée pour tout assembler et demander : « Si la valeur de la colonne prenom est Anne, montre-moi l'enregistrement. »

WHERE dit à votre logiciel que vous voulez voir quelque chose de précis.

Ajoutez ceci à votre WHERE et cela dit que vous voulez chercher uniquement dans les valeurs de la colonne appelée prenom.

= est du langage SQL pour « est ».

La valeur pour votre colonne prenom. N'oubliez pas d'utiliser des guillemets simples pour les chaînes de caractères.

La console ci-dessous vous montre les lignes qui ont été renvoyées par cette requête, où le prénom est égal à Anne.

```
Fichier Edition Fenêtre Aide PasDeRendezVous
> SELECT * FROM mes_contacts WHERE prenom = 'Anne';
+-----------+---------+---------------------------+-----+-------------+--------------+
| nom       | prenom  | email                     | sexe | anniversaire| lieu         |
+-----------+---------+---------------------------+-----+-------------+--------------+
| Toge      | Anne    | Anne_Toge@limosbrillantes.com| F  | NULL        | St-Malo, 35  |
| Lavilliers | Anne   | al86@objectville.net      | F   | NULL        | Strasbourg, 67|
| Hardy     | Anne    | anneh@basenhaut.com       | F   | NULL        | St-Malo, 35  |
| Dubois    | Anne    | anned@cafestarbuzz.com    | F   | NULL        | St-Malo, 35  |
| Sardou    | Anne    | annesardou@pizzaprestissima.com | F | NULL      | St-Malo, 35  |
| Jacob     | Anne    | anne99@objectville.net    | F   | NULL        | Rennes, 35   |
+-----------+---------+---------------------------+-----+-------------+--------------+
6 rows in set (3.67 sec)
```

Voici les résultats de notre instruction SELECT.

Attendez une seconde, vous n'allez pas passer ce * en douce devant mes yeux. Que fait-il exactement ?

Qu'est-ce que c'est que ce * ?

Cette étoile dit au SGBDR de renvoyer les valeurs de **toutes** les colonnes de votre table.

```
SELECT * FROM mes_contacts
WHERE prenom = 'Anne';
```

Je suis une étoile !

Quand vous voyez SELECT *, pensez : demander au logiciel SQL de SÉLECTIONNER TOUTES LES COLONNES.

Utilisez l'étoile pour sélectionner toutes les colonnes de votre table.

il n'y a pas de

Questions stupides

Q: Alors si je ne veux pas sélectionner toutes les colonnes ? Je peux utiliser autre chose que l'étoile ?

R: En fait oui, vous pouvez. L'étoile sélectionne tout, mais dans quelques pages vous apprendrez à ne sélectionner que certaines colonnes pour rendre vos résultats plus faciles à interpréter.

Q: Est-ce que cette étoile est la même chose qu'un astérisque ?

R: Oui, c'est le même caractère sur votre clavier, juste à gauche de la touche Entrée. C'est la même pour les utilisateurs de Mac et de PC.

Cependant, même si c'est exactement le même caractère que l'astérisque, en jargon SQL, on parle toujours d'*étoile*. Cela vaut mieux parce que dire « `SELECT astérisque FROM ...` » n'est pas aussi facile que « `SELECT étoile FROM ...` »

Q: Est-ce qu'il y a d'autres caractères qui ont une signification spéciale, comme l'étoile ?

R: SQL a effectivement d'autres caractères spéciaux ou réservés. Vous allez en voir d'autres dans le livre, un peu plus tard, mais l'étoile est le seul que vous avez besoin de connaître pour l'instant. C'est le seul utilisé dans la partie SELECT d'une instruction SQL.

Le Bar Tête la première ajoute des boissons à base de fruits à son menu. En utilisant ce que vous avez appris dans le premier chapitre, créez la table de cette page et insérez-y les données présentées.

Cette table fait partie d'une base de données appelée **boissons**. Elle contient la table **boissons_faciles** avec des recettes de boissons qui n'ont que deux ingrédients.

boissons_faciles

nom_boisson	principal	qte1	second	qte2	instructions
Épine noire	Schweppes	1.5	jus d'ananas	1	mélangez avec de la glace pilée, filtrez dans un verre à cocktail avec un ruban d'écorce de citron
Lune bleue	soda	1.5	jus de cassis	.75	mélangez avec de la glace pilée, filtrez dans un verre à cocktail avec un ruban d'écorce de citron
Oh mon Dieu	nectar de pêche	1	jus d'ananas	1	mélangez avec de la glace, filtrez dans un verre à liqueur
Tourbillon de citron vert	Sprite	1.5	jus de citron vert	.75	mélangez avec de la glace pilée, filtrez dans un verre à cocktail
Baiser sur les lèvres	jus de cerise	2	nectar d'abricot	7	servez sur de la glace avec une paille
Or brûlant	nectar de pêche	3	jus d'orange	6	versez le jus d'orange chaud dans une grande tasse et ajoutez le nectar de pêche
Arbre solitaire	soda	1.5	jus de cerise	.75	mélangez avec de la glace pilée, filtrez dans un verre à cocktail
Lévrier	soda	1.5	jus de pamplemousse	5	servez sur de la glace pilée, mélangez bien
Été indien	jus de pomme	2	thé chaud	6	versez le jus dans une grande tasse et recouvrez avec le thé chaud
Grenouille dingue	thé glacé	1.5	limonade	5	servez sur des glaçons avec une rondelle de citron vert
Soda et Ça	soda	2	jus de raisin	1	secouez dans un verre à cocktail, pas de glace

Réponses page 118.

qte1 et qte2 sont mesurées en verres de 250 ml.

Ouvrez les yeux !

Prévoyez avant de commencer.

Choisissez vos types de données avec soin et n'oubliez pas NULL. Faites très attention aux apostrophes : c'est un caractère réservé SQL. Pour indiquer à votre SGBDR qu'elle fait partie de votre chaîne de caractère, vous devez placer un antislash devant l'apostrophe (plus d'infos p.68). Puis ***vérifiez votre code à la page 59****.*

À vos crayons !

Ne vous inquiétez pas des caractères des requêtes que vous n'avez pas encore vus. Tapez-les simplement comme vous les voyez pour l'instant et voyez si cela fonctionne.

NOMMEZ CETTE BOISSON

Utilisez la table **boissons_faciles** que vous venez de créer et essayez ces requêtes sur votre machine. Notez quelles boissons sont renvoyées comme résultats pour chaque requête.

```
SELECT * FROM boissons_faciles WHERE principal = 'Sprite';
```

Quelle(s) boisson(s) ? .....

```
SELECT * FROM boissons_faciles WHERE principal = soda;
```

Quelle(s) boisson(s) ? .....

```
SELECT * FROM boissons_faciles WHERE qte2 = 6;
```

Quelle(s) boisson(s) ? .....

```
SELECT * FROM boissons_faciles WHERE second = "jus d\'orange";
```

Quelle(s) boisson(s) ? .....

```
SELECT * FROM boissons_faciles WHERE qte1 < 1.5;
```

Quelle(s) boisson(s) ? .....

```
SELECT * FROM boissons_faciles WHERE qte2 < '1';
```

Quelle(s) boisson(s) ? .....

```
SELECT * FROM boissons_faciles WHERE principal > 'soda';
```

Quelle(s) boisson(s) ? .....

```
SELECT * FROM boissons_faciles WHERE qte1 = '1.5';
```

Quelle(s) boisson(s) ? .....

Oui, vous avez tout à fait raison.

L'une de ces requêtes ne fonctionnera pas. Les autres fonctionnent mais les résultats de certaines ne sont pas ce à quoi vous pourriez vous attendre.

Pour des points de bonus, notez ici quelle requête ne fonctionne pas…

. .....

… et lesquelles fonctionnent alors que vous ne vous y attendiez pas.

. .....

. .....

. .....

. .....

À vos crayons ! Solution

NOMMEZ CETTE BOISSON

Vous avez essayé ces requêtes dans votre table **boissons_faciles** et noté quelles boissons sont renvoyées comme résultat pour chaque requête.

```
SELECT * FROM boissons_faciles WHERE principal = 'Sprite';
```

Quelle(s) boisson(s) ? Tourbillon de citron vert

Remarquez les guillemets simples.

```
SELECT * FROM boissons_faciles WHERE principal = soda;
```

Quelle(s) boisson(s) ? Error

Hum. On dirait que voilà la requête qui ne fonctionne pas.

```
SELECT * FROM boissons_faciles WHERE qte2 = 6;
```

Celle-ci fonctionne. C'est une variable DEC, il n'y a donc besoin d'aucun guillemet.

Quelle(s) boisson(s) ? Or brûlant, Été indien

```
SELECT * FROM boissons_faciles WHERE second = "jus d\'orange";
```

Quelle(s) boisson(s) ? Or brûlant

```
SELECT * FROM boissons_faciles WHERE qte1 < 1.5;
```

Quelle(s) boisson(s) ? Oh mon Dieu

```
SELECT * FROM boissons_faciles WHERE qte2 < '1';
```

Quelle(s) boisson(s) ? Lune bleue, Tourbillon de citron vert, Arbre solitaire

```
SELECT * FROM boissons_faciles WHERE principal > 'soda';
```

Une autre clause WHERE bien écrite.

Quelle(s) boisson(s) ? Grenouille dingue, Tourbillon de citron vert

```
SELECT * FROM boissons_faciles WHERE qte1 = '1.5';
```

Quelle(s) boisson(s) ? Épine noire, Lune bleue, Tourbillon de citron vert, Arbre solitaire, Lévrier, Grenouille dingue

Pour des points de bonus, notez ici quelle requête ne fonctionne pas…

C'est la clause WHERE qui ne va pas fonctionner. Il faut des guillemets autour de cette VARCHAR.

```
WHERE principal = soda;
```

… et lesquelles fonctionnent alors que vous ne vous y attendiez pas.

```
WHERE second = "jus d\'orange";
```

Cette requête fonctionne et ne cause pas d'erreur mais il est déconseillé d'utiliser des guillemets doubles. Ceux-ci sont utilisés en PHP pour signaler une requête SQL.

```
WHERE qte2 < '1';
```

Mais celle-ci fonctionne, même si elle ne devrait pas parce que les variables DEC n'ont pas besoin de guillemets.

```
WHERE qte1 = '1.5';
```

Et pareil pour celle-ci !

Ces deux dernières requêtes vont fonctionner parce que la plupart des SGBDR SQL vous laissent un peu de lattitude. Ils vont ignorer les guillemets et traiter vos valeurs DEC et INT en tant que nombres, même si les guillemets indiquent que ce sont des valeurs texte. Les requêtes ne sont PAS CORRECTES mais votre SGBDR est accomodant.

Comment interroger vos types de données

Pour écrire des clauses WHERE valides, vous devez vérifier que chacun des types de données que vous utilisez est formaté correctement. Voici les conventions pour les types de données les plus importants :

Nous ♥ les guillemets simples	Pas de guillemets pour nous
CHAR	DEC
VARCHAR	INT
DATE	
DATETIME, TIME ou TIMESTAMP	
BLOB	

Il faut des guillemets simples pour les types de données VARCHAR, CHAR, , DATE et TIME. Rien pour les types numériques, DEC et INT.

Des problèmes de ponctuation supplémentaires

Greg a obtenu quelques contacts de plus dans une soirée. Il essaie d'en ajouter un à sa table :

Stéphane Dropeau
Anniv : 1/4/1970
Punk

Célibataire

Moulin, 03
Stephane@pistaches.com

Centres d'intérêt : écraser l'état

Cherche : compatriotes, joueurs de guitare

```
INSERT INTO mes_contacts
VALUES
('Dropeau','Stéphane','stephane@pistaches.com',
'H','1970-04-01','Punk','Moulin, 03',
'Célibataire','écraser l'État','compatriotes,
joueurs de guitare');
```

Mais on dirait que son programme ne répond pas. Il tape quelques points-virgules pour essayer de terminer la requête. Pas de résultat.

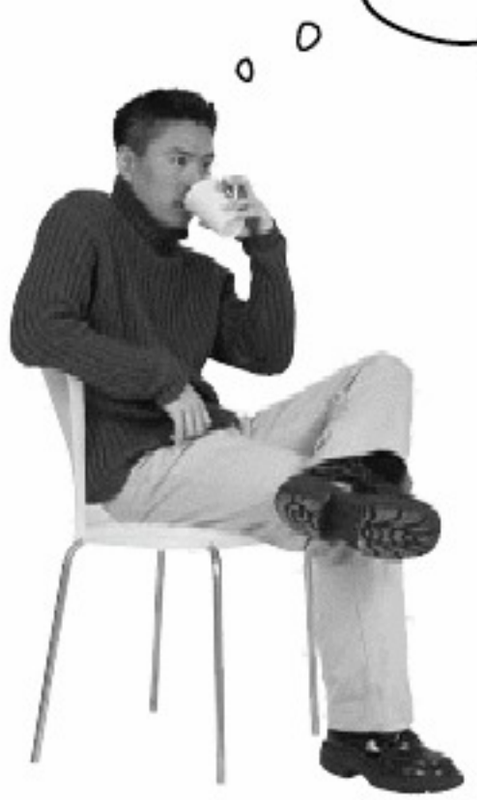
```
Fichier Edition Fenêtre Aide ExtraTerrestres !
> INSERT INTO mes_contacts VALUES ('Dropeau', 'Stéphane',
'stephane@pistaches.com', 'H','1970-04-01', 'Punk', 'Moulin,
03', 'Célibataire', 'écraser l'État', 'compatriotes, joueurs
de guitare');
'>
'> ;
'> ;
'>
```

À chaque fois qu'il appuie sur Entrée, il voit ces signes : '>

MUSCLEZ VOS NEURONES

Que se passe-t-il ici selon vous ?

Hmm, regardez ce guillemet simple qui apparaît devant le > à chaque fois. Je parie qu'il y a un problème avec les guillemets dans notre instruction INSERT…



Guillemets simples dépareillés

Exactement ! Quand Greg a essayé d'ajouter l'enregistrement, le programme SQL attendait un nombre pair de guillemets simples, un devant et un après chaque valeur VARCHAR, CHAR et DATE. Le centre d'intérêt, **écraser l'État**, pose problème parce qu'il contient une apostrophe en plus. Le SGBDR SQL attend toujours un guillemet simple de plus.

Vous pouvez reprendre le contrôle de votre console.

Terminez l'instruction en tapant un guillemet simple et un point-virgule. Cela donne au SGBDR le guillemet simple qu'il attend.

Vous obtenez une erreur quand vous tapez l'autre guillemet et le point-virgule et vous devez recommencer à entrer votre **INSERT** depuis le début.

Vous obtenez une erreur en faisant cela mais au moins vous pouvez réessayer.

Fichier Edition Fenêtre Aide IlEnFautDeux

```
> INSERT INTO mes_contacts VALUES ('Dropeau', 'Stéphane',
'stephane@pistaches.com', 'H','1970-04-01', 'Punk', 'Moulin,
03', 'Célibataire', 'écraser l'État', 'compatriotes, joueurs
de guitare');

'>

'> ;

'> ;

'>

'> ';

ERROR 1064 (42000): You have an error in your SQL syntax;
check the manual that corresponds to your SQL server version
for the right syntax to use near 'État', 'compatriotes,
joueurs de guitare');

' at line 1

>
```

Taper un guillemet simple et un point-virgule termine l'instruction INSERT cassée.

Cette erreur vous donne une idée assez claire de ce qui ne va pas. Elle cite une partie de votre requête en commençant par le guillemet simple de trop.

Même si l'enregistrement n'est pas insère, ce dernier > montre qu'au moins le programme SQL répond de nouveau.

Les guillemets simples sont des caractères spéciaux

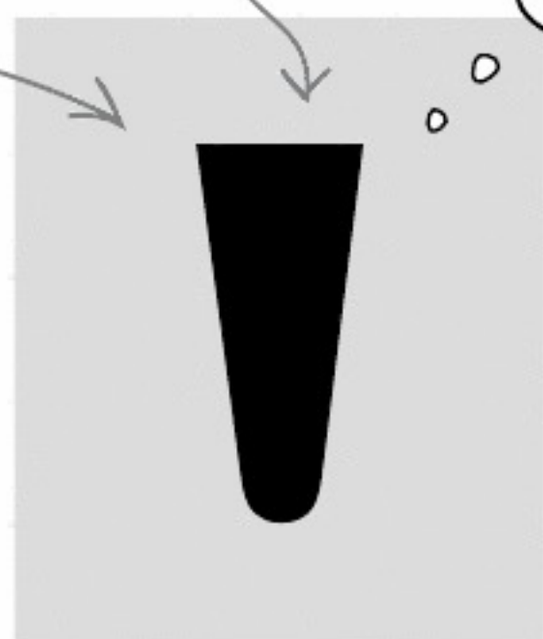
Quand vous essayez d'insérer un VARCHAR, CHAR ou BLOB contenant une apostrophe, vous devez indiquer à votre SGBDR qu'elle n'est pas là pour terminer le texte ***mais qu'elle en fait partie*** et doit être incluse dans la ligne. On peut le faire en **ajoutant un antislash** devant le guillemet simple.

```
INSERT INTO mes_contacts
(centres_interet)
VALUES
('écraser l\'État');
```

Quand je suis seul, j'ai besoin d'un antislash.

Le guillemet simple est un caractère SQL « réservé ». Cela signifie qu'il a une fonction spécifique dans le langage.

On l'utilise pour dire au logiciel SQL où une chaîne de caractères commence et finit.



il n'y a pas de Questions stupides

Q: N'est-ce pas la même chose qu'une apostrophe ?

R: C'est exactement la même chose qu'une apostrophe. Mais SQL lui assigne une signification très particulière. Il est utilisé pour dire au logiciel SQL que les données entre deux de ces signes sont des données texte.

Q: Quels types de données ont en besoin ?

R: Les types de données textuelles. Les données texte signifient simplement que les données sont une colonne VARCHAR, CHAR, BLOB ou TIMEDATE. Tout ce qui n'est pas un nombre.

Q: Est-ce que les colonnes DEC et INT en ont besoin ?

R: Non. Les colonnes numériques ne contiennent pas d'espaces, il est donc facile de dire quand un nombre se termine et quand commence le mot suivant de l'instruction.

Q: Donc, ce n'est utilisé que pour les colonnes de texte ?

R: Oui. Le seul problème est que les colonnes de texte contiennent des espaces. Cela pose des problèmes quand vos données contiennent des apostrophes. SQL ne peut pas déterminer la différence entre une apostrophe à l'intérieur de la colonne et un guillemet simple qui lui dit quand la colonne commence ou finit.

Q: Est-ce que cela ne serait pas plus facile de les distinguer si on utilisait des guillemets doubles plutôt que simples ?

R: Non. N'utilisez pas de guillemets doubles au cas où vous utiliseriez des instructions SQL avec un langage de programmation (comme PHP) plus tard. Vous utilisez '' dans le langage de programmation pour dire « c'est là que se trouve l'instruction SQL » ; ainsi, les guillemets simples sont reconnus comme appartenant à cette instruction et non pas au langage de programmation.

INSERTion de données contenant des apostrophes

Vous devez dire à votre logiciel SQL que votre apostrophe n'est pas là pour commencer ou finir une chaîne de caractères, mais fait simplement *partie de* la chaîne de caractères.

Faire passer les apostrophes avec un antislash

Vous pouvez le faire (et réparer votre instruction `INSERT` par la même occasion) en ajoutant un signe antislash devant l'apostrophe de votre chaîne de caractères :

```
INSERT INTO mes_contacts
VALUES
('Dropeau','Stéphane','stephane@pistaches.com', 'H',
'1970-04-01', 'Punk','Moulin, 03','Célibataire',
'écraser l\'état','compatriotes, joueurs de
guitare');
```

Dire à SQL qu'un guillemet simple fait partie d'une chaîne de caractères en plaçant un antislash devant s'appelle y « échapper ».

Faire passer les apostrophes avec un guillemet simple en plus

Une autre façon d' « échapper » à l'apostrophe est de placer un guillemet en plus devant elle.

```
INSERT INTO mes_contacts
VALUES
('Dropeau','Stéphane','stephane@pistaches.com', 'H',
'1970-04-01', 'Punk','Moulin, 03','Célibataire',
'écraser l''État','compatriotes, joueurs de
guitare');
```

Ou vous pouvez « échapper » à une apostrophe en plaçant un guillemet simple devant elle.

Quels autres caractères pourraient poser des problèmes semblables ?

Si vous avez des données avec des guillemets dans votre table, il est possible que vous deviez les rechercher avec une clause WHERE à un moment. Pour SELECTionner les données contenant des guillemets simples dans votre clause WHERE, vous devez échapper à votre guillemet simple de la même façon que quand vous l'avez inséré.

Réécrivez le code ci-dessous en utilisant différentes méthodes pour échapper au guillemet simple.

```
SELECT * FROM mes_contacts
WHERE
centres_interet = 'écraser l'État';
```

1 .....

.....

.....

2 .....

.....

.....

Quelle méthode préférez-vous ?

Si vous avez des données avec des guillemets dans votre table, il est possible que vous deviez les rechercher avec une clause WHERE à un moment. Pour SELECTionner les données contenant des guillemets simples dans votre clause WHERE, vous devez échapper à votre guillemet simple, de la même façon que quand vous l'avez inséré.

Réécrivez le code ci-dessous en utilisant différentes méthodes pour échapper au guillemet simple.

```
SELECT * FROM mes_contacts
WHERE
centres_interet = 'écraser l'État';
```

1

```
SELECT * FROM mes_contacts
WHERE
centres_interet = 'écraser l\État';
```

Méthode 1, l'antislash.

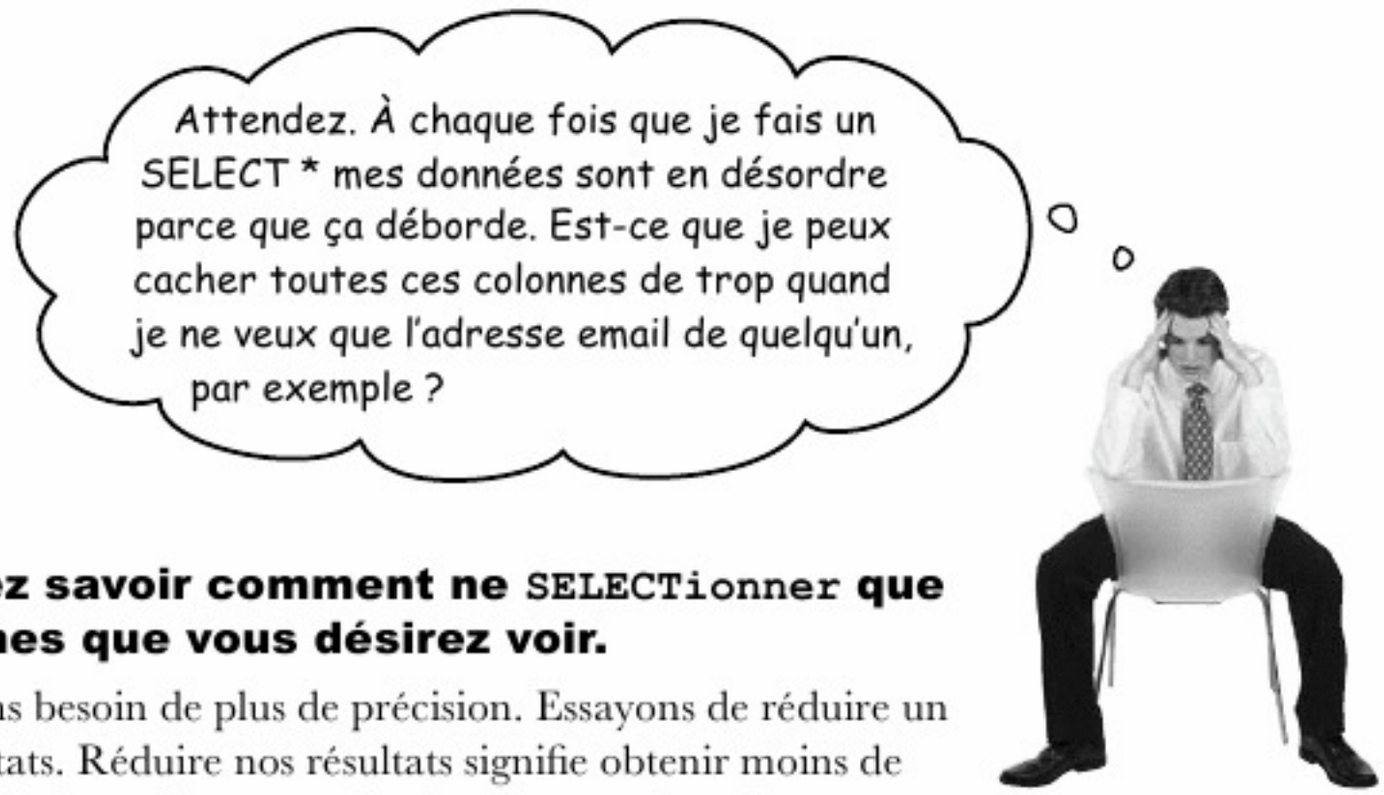
2

```
SELECT * FROM mes_contacts
WHERE
centres_interet = 'écraser l''État';
```

Méthode 2, le guillemet simple en plus.

SELECTionner des données précises

Maintenant vous maîtrisez la façon de `SELECTionner` tous les types de données avec des guillemets et comment `SELECTionner` les données contenant des guillemets.



Vous devez savoir comment ne `SELECTionner` que les colonnes que vous désirez voir.

Ici, nous avons besoin de plus de précision. Essayons de réduire un peu nos résultats. Réduire nos résultats signifie obtenir moins de colonnes à l'affichage. Nous ne sélectionnons que les colonnes que nous voulons voir.

À faire à la maison

Solutions des exercices

Avant d'essayer cette requête SELECT, dessinez les grandes lignes de la table des résultats telle que vous l'imaginez.
(Si vous avez besoin de consulter la table boissons_faciles, vous pouvez la trouver page 59.)

Nous avons remplacé l'* avec ces noms de colonnes.

```
SELECT nom_boisson, principal, second
FROM boissons_faciles
WHERE principal = 'soda';
```

À faire à la maison

Solutions des exercices.

Avant d'essayer cette requête SELECT, dessinez les grandes lignes de la table des résultats telle que vous l'imaginez.

nom_boisson	principal	second
Lune bleue	soda	jus de cassis
Arbre solitaire	soda	jus de cerise
Lévrier	soda	jus de pamplemousse
Soda et Ça	soda	jus de raisin

À l'ancienne

```
SELECT * FROM boissons_faciles;
```

Ici nous avons toutes les colonnes et nos résultats sont trop larges pour la fenêtre de notre terminal. Ils débordent sur la ligne suivante et l'affichage est en désordre.

Fichier Edition Fenêtre Aide AffichageDésordonné

```
> SELECT * FROM boissons_faciles;
+-------------------------+----------------+-----+---------------------+-----+---------
----------------------------------------------------------------------------------------+
| nom_boisson             | principal      | qte1 | second              | qte2 |
instructions                                                                            |
+-------------------------+----------------+-----+---------------------+-----+---------
----------------------------------------------------------------------------------------+
| Baiser sur les lèvres   | jus de cerise  |  2.0 | nectar d'abricot    | 7.00 | servez
sur de la glace avec une paille                                                         |
| Or brûlant              | nectar de pêche |  3.0 | jus d'orange        | 6.00 | versez
le jus d'orange chaud dans une grande tasse et ajoutez le nectar de pêche               |
| Arbre solitaire         | soda           |  1.5 | jus de cerise       | 0.75 | mélangez
avec de la glace pilée, filtrez dans un verre à cocktail                                |
| Lévrier                 | soda           |  1.5 | jus de pamplemousse | 5.00 | servez
sur de la glace pilée, mélangez bien                                                    |
| Été indien              | jus de pomme   |  2.0 | thé chaud           | 6.00 | versez
le jus dans une grande tasse et recouvrez avec le thé chaud                             |
| Grenouille dingue       | thé glacé      |  1.5 | limonade            | 5.00 | servez
sur des glaçons avec une rondelle de citron vert                                        |
| Soda et Ça              | soda           |  2.0 | jus de raisin       | 1.00 | secouez
dans un verre à cocktail, pas de glace                                                  |
| Épine noire             | Schweppes      |  1.5 | jus d'ananas        | 1.00 | mélangez
avec de la glace pilée, filtrez dans un verre à cocktail avec un ruban d'écorce de citron |
| Lune bleue              | soda           |  1.5 | jus de cassis       | 0.75 | mélangez
avec de la glace pilée, filtrez dans un verre à cocktail avec un ruban d'écorce de citron |
| Oh mon Dieu             | nectar de pêche |  1.0 | jus d'ananas        | 1.00 | mélangez
avec de la glace, filtrez dans un verre à liqueur                                       |
| Tourbillon de citron vert| Sprite         |  1.5 | jus de citron vert  | 0.75 | mélangez
avec de la glace pilée, filtrez dans un verre à cocktail                                |
+-------------------------+----------------+-----+---------------------+-----+---------
----------------------------------------------------------------------------------------+
11 rows in set (0.00 sec)
```

SELECTionner des colonnes spécifiques pour limiter les résultats

En précisant les colonnes que nous voulons que notre requête renvoie, nous pouvons choisir uniquement les valeurs des colonnes dont nous avons besoin. Tout comme vous utilisez une clause WHERE pour limiter le nombre de lignes, vous pouvez utiliser une sélection de colonnes pour limiter le nombre de colonnes. L'intérêt est de laisser SQL faire le sale boulot pour vous.

```
SELECT nom_boisson, principal, second
FROM boissons_faciles;
```

...mais nous pouvons réduire nos résultats en ne sélectionnant que les colonnes que nous voulons voir affichées dans les résultats.

Fichier Edition Fenêtre Aide JusteCeQuilFaut

```
> SELECT nom_boisson, principal, second FROM boissons_faciles;
+--------------------------+----------------+--------------------+
| nom_boisson              | principal      | second             |
+--------------------------+----------------+--------------------+
| Baiser sur les lèvres    | jus de cerise  | nectar d'abricot   |
| Or brûlant               | nectar de pêche | jus d'orange      |
| Arbre solitaire          | soda           | jus de cerise      |
| Lévrier                  | soda           | jus de pamplemousse |
| Été indien               | jus de pomme   | thé chaud          |
| Grenouille dingue        | thé glacé      | limonade           |
| Soda et Ça               | soda           | jus de raisin      |
| Épine noire              | Schweppes      | jus d'ananas       |
| Lune bleue               | soda           | jus de cassis      |
| Oh mon Dieu              | nectar de pêche | jus d'ananas      |
| Tourbillon de citron vert | Sprite        | jus de citron vert |
+--------------------------+----------------+--------------------+
11 rows in set (0.00 sec)
```

SELECTionner des colonnes spécifiques pour des résultats plus rapides

C'est une bonne habitude de programmation et, de plus, cela présente d'autres avantages. Si vos tables deviennent plus larges, cela accélère le renvoi de vos résultats. Ce sera aussi plus rapide si vous utilisez SQL avec un autre langage de programmation comme PHP.

À vos crayons !

Diverses manières d'obtenir un Baiser sur les lèvres

Vous vous souvenez de notre table boissons_faciles ? Cette instruction SELECT aura pour résultat un Baiser sur les lèvres :

```
SELECT nom_boisson FROM boissons_faciles
WHERE
principal = 'jus de cerise';
```

Complétez les quatre autres instructions SELECT page suivante pour obtenir d'autres Baisers.

boissons_faciles

nom_boisson	principal	qte1	second	qte2	instructions
Épine noire	Schweppes	1.5	jus d'ananas	1	mélangez avec de la glace pilée, filtrez dans un verre à cocktail avec un ruban d'écorce de citron
Lune bleue	soda	1.5	jus de cassis	.75	mélangez avec de la glace pilée, filtrez dans un verre à cocktail avec un ruban d'écorce de citron
Oh mon Dieu	nectar de pêche	1	jus d'ananas	1	mélangez avec de la glace, filtrez dans un verre à liqueur
Tourbillon de citron vert	Sprite	1.5	jus de citron vert	.75	mélangez avec de la glace pilée, filtrez dans un verre à cocktail
Baiser sur les lèvres	jus de cerise	2	nectar d'abricot	7	servez sur de la glace avec une paille
Or brûlant	nectar de pêche	3	jus d'orange	6	versez le jus d'orange chaud dans une grande tasse et ajoutez le nectar de pêche
Arbre solitaire	soda	1.5	jus de cerise	.75	mélangez avec de la glace pilée, filtrez dans un verre à cocktail
Lévrier	soda	1.5	jus de pamplemousse	5	servez sur de la glace pilée, mélangez bien
Été indien	jus de pomme	2	thé chaud	6	versez le jus dans une grande tasse et recouvrez avec le thé chaud
Grenouille dingue	thé glacé	1.5	limonade	5	servez sur des glaçons avec une rondelle de citron vert
Soda et Ça	soda	2	jus de raisin	1	secouez dans un verre à cocktail, pas de glace

```
SELECT .....
WHERE  .....

SELECT .....
WHERE  .....

SELECT .....
WHERE  .....

SELECT .....
WHERE  .....
```

Écrivez maintenant trois instructions SELECT qui vous donneront une Grenouille dingue.

1. .....
   .....
   .....

2. .....
   .....

3. .....
   .....

À vos crayons ! Solution

Complétez les quatre autres instructions SELECT pour obtenir d'autres Baisers.

```
SELECT    nom_boisson FROM boissons_faciles
WHERE     second = 'nectar d\'abricot';
```

```
SELECT    nom_boisson FROM boissons_faciles
WHERE     qte2 = 7;
```

```
SELECT    nom_boisson FROM boissons_faciles
WHERE     instructions = 'servez sur de la glace avec une paille';
```

```
SELECT    nom_boisson FROM boissons_faciles
WHERE     nom_boisson = 'Baiser sur les lèvres';
```

En voilà une que vous utiliserez rarement mais elle vous donne bien sûr le résultat voulu. Vous pouvez utiliser quelque chose comme cela quand vous voulez vérifier que votre colonne nom_boisson n'a pas de problème de typo.

Écrivez maintenant trois instructions SELECT qui vous donneront une Grenouille dingue.

1. SELECT nom_boisson FROM boissons_faciles
   WHERE principal = 'thé glacé';

2. SELECT nom_boisson FROM boissons_faciles
   WHERE second = 'limonade';

3. SELECT nom_boisson FROM boissons_faciles
   WHERE instructions = 'servez sur des glaçons avec une rondelle de citron vert';

POINTS D'IMPACT

- Utilisez des guillemets simples dans votre clause WHERE quand vous sélectionnez des champs de texte.
- N'utilisez pas de guillemets simples quand vous sélectionnez des champs numériques.
- Utilisez l'* dans votre SELECT si vous voulez sélectionner toutes les colonnes.
- Si vous avez entré votre requête et que votre SGBDR n'en finit pas de l'exécuter, vérifiez s'il manque un guillemet simple.
- Si vous le pouvez, sélectionnez des colonnes spécifiques dans votre table plutôt que d'utiliser SELECT *.

il n'y a pas de

Questions stupides

Q: Alors si j'ai besoin que toutes les colonnes de ma table soient renvoyées par une requête ? Est-ce que je dois vraiment les écrire dans le SELECT plutôt que d'utiliser l' * ?

R: Si vous avez besoin de toutes les colonnes, alors surtout utilisez l' *. Ce n'est que lorsque vous n'avez pas besoin de tout que vous devez essayer de ne pas l'utiliser.

Q: J'ai essayé de copier-coller une requête à partir d'internet et j'obtiens toujours des erreurs quand j'essaie de l'utiliser. Est-ce que je fais mal quelque chose ?

R: Les requêtes collées à partir des navigateurs web contiennent parfois des caractères invisibles qui ressemblent à des espaces mais signifient quelque chose d'autre pour SQL. Les coller dans un éditeur de texte permet de voir et d'enlever ces caractères "gremlins". La meilleure solution est de les coller dans un éditeur de texte pour les regarder de plus près.

Q: Alors je dois la coller dans quelque chose comme Microsoft Word ?

R: Non, Word n'est pas un très bon choix parce qu'il ne vous montrera que le formattage invisible qui peut se trouver dans le texte. Essayez le Bloc-notes (PC) ou TextEdit en mode texte (Mac).

Q: Pour échapper à l'apostrophe, existe-t-il des raisons d'utiliser une méthode plutôt qu'une autre ?

R: Pas vraiment. Nous avons tendance à utiliser la méthode antislash simplement parce que nous trouvons qu'il est plus facile de repérer l'apostrophe en plus quand une requête tourne mal. Par exemple, il est plus facile d'examiner visuellement :

```
'N\'est-ce pas l\'ardoise de ta soeur ?'
```

Que cela :

```
'N''est-ce pas l''ardoise de ta soeur ?'
```

À part cela, il n'y a vraiment aucune raison de préférer une méthode à l'autre. Les deux méthodes vous permettent d'entrer des apostrophes dans vos colonnes de texte.

La crêpe demande ce que votre table peut faire pour elle...

Pour trouver la meilleure crêpe au sucre dans la table, il vous faut au moins deux instructions `SELECT`. La première va sélectionner les lignes avec le bon type de crêpe. La deuxième va sélectionner les lignes de crêpes avec une note de 10.

Je veux trouver la meilleure crêpe au sucre sans devoir fouiller dans tous ces résultats.

evaluation_crepes

lieu	heure	date	type	note	commentaires
Café Starbuzz	7:43	23/4	sucre canelle	6	trop d'épices
Crêperie du port	8:56	25/8	sucre	5	grasse
Crêperie du port	19:58	26/4	fourrée	6	pas très chaude mais savoureuse
Café Starbuzz	22:35	24/4	sucre	7	tiède mais pas chaude
Le roi de la croustille	21:39	26/9	fourrée	6	manque de confiture
Café Starbuzz	7:48	23/4	éclair	10	chocolat !
Le roi de la croustille	8:56	25/11	sucre	8	avec sirop d'érable

Imaginez que cette table contient 10,000 enregistrements.

1 On peut chercher le type de crêpe :

Vous devez SELECTionner la note pour chercher dans les meilleurs scores et le lieu parce qu'il vous donne le nom du vainqueur.

```
SELECT lieu, note FROM evaluation_crepes
WHERE
type = 'sucre';
```

Tous les résultats auront le bon type de crêpe.

Résultats de la première requête, mais imaginez-en des centaines d'autres.

lieu	note
Crêperie du port	5
Café Starbuzz	7
Le roi de la croustille	8
Café Starbuzz	10
Crêperie du port	8

Demandez-vous ce que vous pouvez faire pour votre crêpe

❷ Vous pouvez aussi chercher la meilleure note :

```
SELECT lieu, type FROM evaluation_crepes
WHERE
note = 10;
```

Vous devez chercher dans tous les types puis le lieu vous donnera le nom de votre vainqueur.

Tous les résultats seront les mieux notés.

lieu	type
Café Starbuzz	éclair
Le roi de la croustille	sucre
Café Starbuzz	sucre
Crêperie du port	éclair

Résultats de la deuxième requête, de même, imaginez-en des centaines de plus.

Cela ne m'aide pas vraiment. Je peux m'arrêter avec l'une ou l'autre requête et chercher dans les résultats mais cette table a des milliers d'enregistrements… J'ai faim et je veux cette crêpe ***maintenant*** !

MUSCLEZ VOS NEURONES

En bon français, à quelle question essayez-vous de répondre avec ces requêtes ?

Combiner vos requêtes

Nous pouvons gérer les deux choses que nous cherchons ('sucre' pour le type et 10 pour la note) en une seule requête en utilisant le mot-clé AND. Les résultats que nous obtenons à cette requête doivent satisfaire aux deux conditions.

```
SELECT lieu
FROM evaluation_crepes
WHERE type = 'sucre'
AND
note = 10;
```

Maintenant il ne reste plus que le lieu à SELECTionner.

Utilisez le mot AND pour combiner vos deux clauses WHERE.

Voici le résultat de la requête AND. Même si nous avions reçu plus d'une ligne comme résultat de notre requête, vous sauriez que tous ces endroits ont des crêpes au sucre avec une note de 10, vous pourriez donc aller dans n'importe lequel d'entre eux. Ou dans tous.

lieu	note
Crêperie du port	5
Café Starbuzz	7
Le roi de la croustille	8
Café Starbuzz	10
Crêperie du port	8

AND

lieu	type
Café Starbuzz	éclair
Le roi de la croustille	sucre
Café Starbuzz	sucre
Crêperie du port	éclair

Cette requête combine les résultats pour type= 'sucre' et note= 10 pour trouver des résultats qui correspondent aux deux requêtes.

lieu
Café Starbuzz

Alors j'aurais pu trouver Anne en utilisant AND ?

En utilisant la table mes_contacts, écrivez des requêtes pour Greg. Ne SELECTionnez que les colonnes dont vous avez vraiment besoin pour donner votre réponse. Faites attention aux guillemets simples.

Écrivez une requête pour trouver les adresses emails de tous les programmeurs.

. .....

. .....

. .....

. .....

Écrivez une requête pour trouver le nom et le lieu de quiconque a la même date de naissance que vous.

. .....

. .....

. .....

. .....

Écrivez une requête pour trouver le nom et l'adresse email de tous les célibataires qui vivent dans votre ville. Pour des points en plus, ne prenez que ceux du sexe que vous voudriez inviter pour un rendez-vous.

. .....

. .....

. .....

. .....

. .....

Écrivez la requête que Greg aurait pu utiliser pour trouver toutes les Anne de Saint-Malo.

. .....

. .....

. .....

. .....

Solutions des exercices

En utilisant la table mes_contacts, écrivez des requêtes pour Greg. Ne SELECTionnez que les colonnes dont vous avez vraiment besoin pour donner votre réponse. Faites attention aux guillemets simples.

Écrivez une requête pour trouver les adresses emails de tous les programmeurs.

```
SELECT email FROM mes_contacts
WHERE profession = 'programmeur';
```

Nous voulons la colonne email.

La profession que nous voulons est programmeur.

Écrivez une requête pour trouver le nom et le lieu de quiconque a la même date de naissance que vous.

```
SELECT nom, prenom, lieu
FROM mes_contacts
WHERE anniversaire = '1975-09-05';
```

Ceci doit être votre date de naissance entre guillemets.

Écrivez une requête pour trouver le nom et l'adresse email de tous les célibataires qui vivent dans votre ville. Pour des points en plus, ne prenez que ceux du sexe que vous voudriez inviter pour un rendez-vous.

```
SELECT nom, prenom, email
FROM mes_contacts
WHERE lieu = 'Nice, 06'
AND sexe = 'H';
```

Votre ville et votre département ici.

Le sexe des personnes avec qui vous voudriez un rendez-vous.

Écrivez la requête que Greg aurait pu utiliser pour trouver toutes les Anne de Saint-Malo.

```
SELECT nom, prenom, email
FROM mes_contacts
WHERE lieu = 'St-Malo, 35'
AND prenom = 'Anne';
```

Si l'on regarde la table, on voit que Greg semble avoir raccourci Saint-Malo en St-Malo. Espérons qu'il l'a fait systématiquement.

Trouver des valeurs numériques

Disons que vous voulez trouver toutes les boissons de la table `boissons_faciles` qui contiennent plus d'un verre de soda en *une seule requête*. Voici la manière compliquée de trouver les résultats. Vous pouvez utiliser deux requêtes :

Nous ne voulons que les noms des boissons.

Les boissons au soda avec 1.5 verres de soda.

```
SELECT nom_boisson FROM boissons_faciles
WHERE
principal = 'soda'
AND
qte1 = 1.5;
```

```
Fichier Edition Fenêtre Aide PlusDeSoda
> SELECT nom_boisson FROM boissons_faciles WHERE principal = 'soda' AND
qte1 = 1.5;
+----------------+
| nom_boisson    |
+----------------+
| Lune bleue     |
| Arbre solitaire |
| Lévrier        |
+----------------+
3 rows in set (0.00 sec)
```

Les boissons au soda avec 2 verres de soda.

```
SELECT nom_boisson FROM boissons_faciles
WHERE
principal = 'soda'
AND
qte1 = 2;
```

```
Fichier Edition Fenêtre Aide EncorePlusDeSoda
> SELECT nom_boisson FROM boissons_faciles WHERE principal =
'soda' AND qte1 = 2;
+------------+
| nom_boisson |
+------------+
| Soda et Ça  |
+------------+
1 row in set (0.00 sec)
```

boissons_faciles

nom_boisson	principal	qte1	second	qte2	instructions
Épine noire	Schweppes	1.5	jus d'ananas	1	mélangez avec de la glace pilée, filtrez dans un verre à cocktail avec un ruban d'écorce de citron
Lune bleue	soda	1.5	jus de cassis	.75	mélangez avec de la glace pilée, filtrez dans un verre à cocktail avec un ruban d'écorce de citron
Oh mon Dieu	nectar de pêche	1	jus d'ananas	1	mélangez avec de la glace, filtrez dans un verre à liqueur
Tourbillon de citron vert	Sprite	1.5	jus de citron vert	.75	mélangez avec de la glace pilée, filtrez dans un verre à cocktail
Baiser sur les lèvres	jus de cerise	2	nectar d'abricot	7	servez sur de la glace avec une paille
Or brûlant	nectar de pêche	3	jus d'orange	6	versez le jus d'orange chaud dans une grande tasse et ajoutez le nectar de pêche
Arbre solitaire	soda	1.5	jus de cerise	.75	mélangez avec de la glace pilée, filtrez dans un verre à cocktail
Lévrier	soda	1.5	jus de pamplemousse	5	servez sur de la glace pilée, mélangez bien
Été indien	jus de pomme	2	thé chaud	6	versez le jus dans une grande tasse et recouvrez avec le thé chaud
Grenouille dingue	thé glacé	1.5	limonade	5	servez sur des glaçons avec une rondelle de citron vert
Soda et Ça	soda	2	jus de raisin	1	secouez dans un verre à cocktail, pas de glace

Une fois suffit

Mais c'est une perte de temps d'utiliser deux requêtes et vous pourriez rater des boissons avec des quantités comme 1.75 ou 3 verres. À la place, vous pouvez utiliser un signe **supérieur** :

```
SELECT nom_boisson FROM boissons_faciles
WHERE
principal = 'soda'
AND
qte1 > 1;
```

Le symbole SUPÉRIEUR vous donnera toutes les boissons qui contiennent plus d'1 verre de soda.

```
Fichier Edition Fenêtre Aide FaitesLeUneFois
> SELECT nom_boisson FROM boissons_faciles  WHERE principal =
'soda' AND qte1 > 1;
+-----------------+
| nom_boisson     |
+-----------------+
| Lune bleue      |
| Arbre solitaire |
| Lévrier         |
| Soda et Ça      |
+-----------------+
4 rows in set (0.00 sec)
```

MUSCLEZ VOS NEURONES

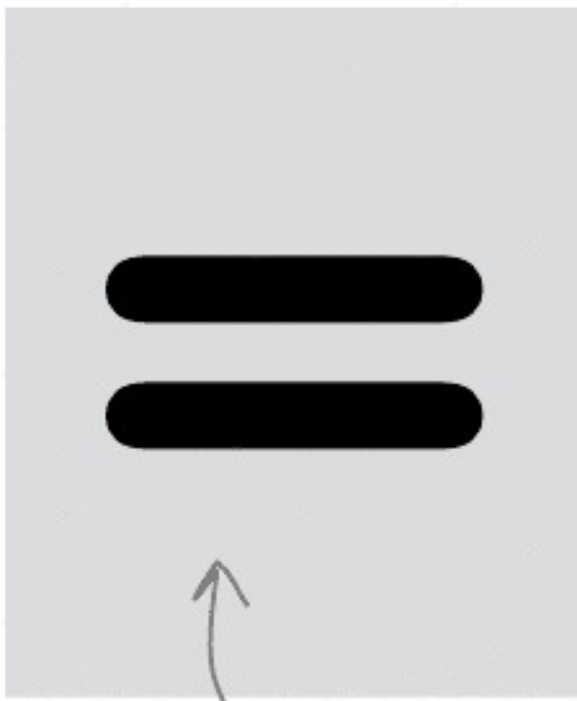
Pourquoi ne pouvez-vous pas combiner les deux premières requêtes avec un AND supplémentaire ?

Opérateurs de comparaison ~~sans histoires~~

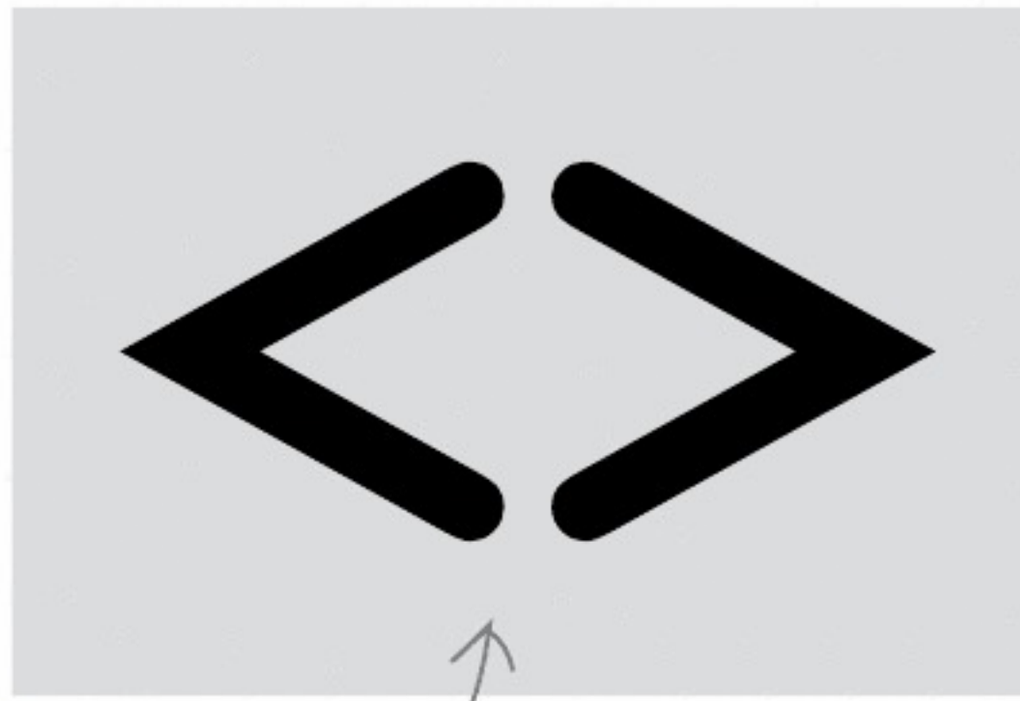
Jusqu'ici, nous avons seulement utilisé le signe **égal** dans notre clause `WHERE`. Vous venez de voir le symbole **supérieur**, >. Il compare une valeur à une autre. Voici le reste des opérateurs de comparaison :

Le signe égal cherche des correspondances exactes. Cela ne nous sert à rien quand nous voulons trouver si quelque chose est inférieur ou supérieur à quelque chose d'autre.

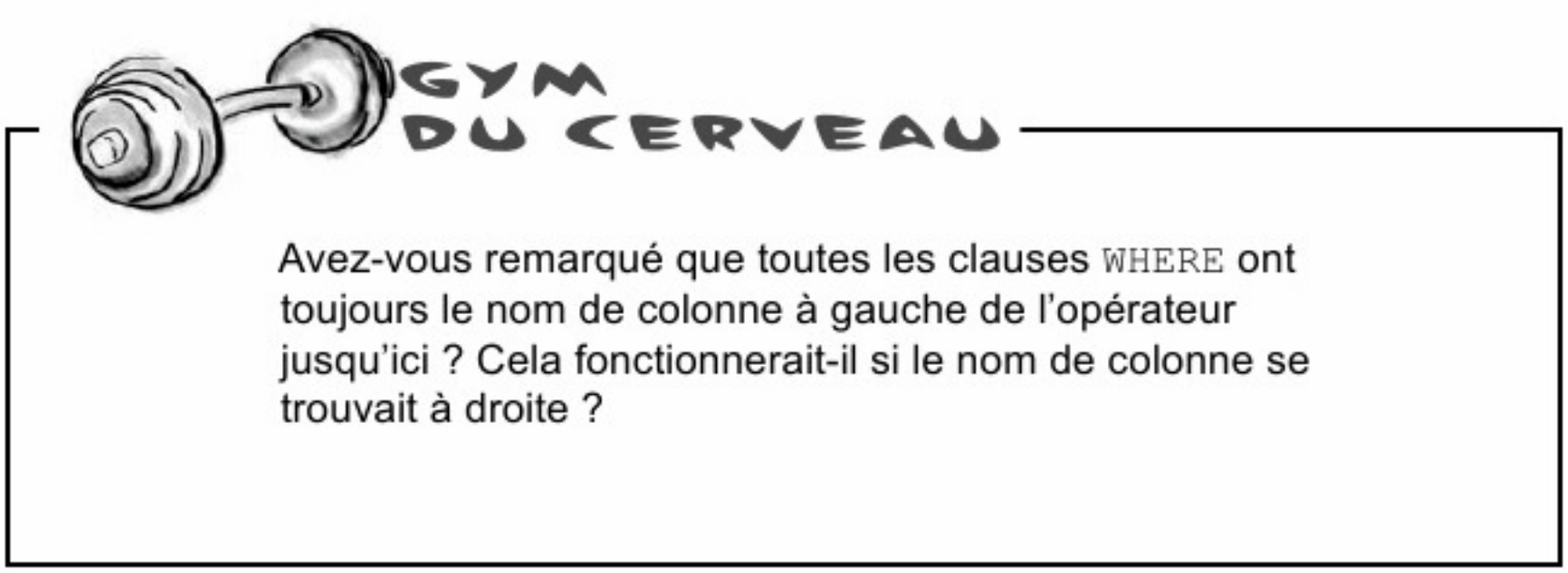
Ce signe déroutant est **différent de**. Il renvoie exactement le résultat opposé du signe égal. Deux valeurs sont, soit égales, soit différentes.



Le signe ÉGAL que nous connaissons et aimons tous.



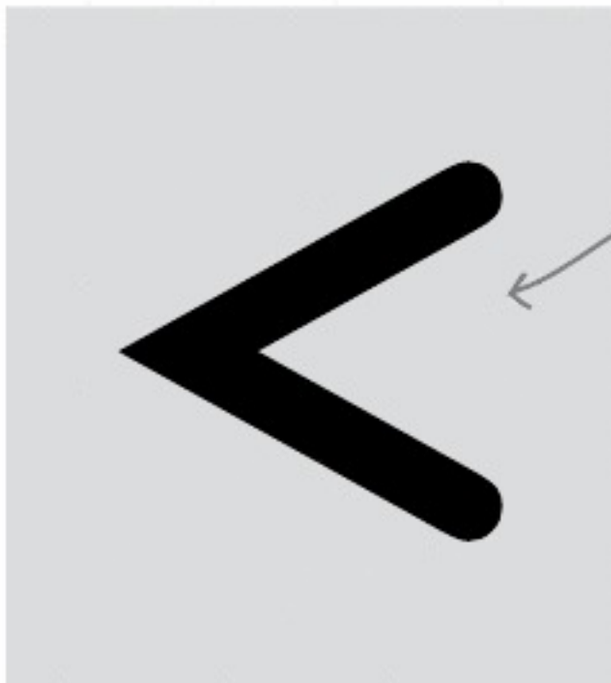
Celui-ci signifie DIFFÉRENT DE. Il renvoie tous les enregistrements qui ne correspondent pas à la condition.



Avez-vous remarqué que toutes les clauses `WHERE` ont toujours le nom de colonne à gauche de l'opérateur jusqu'ici ? Cela fonctionnerait-il si le nom de colonne se trouvait à droite ?

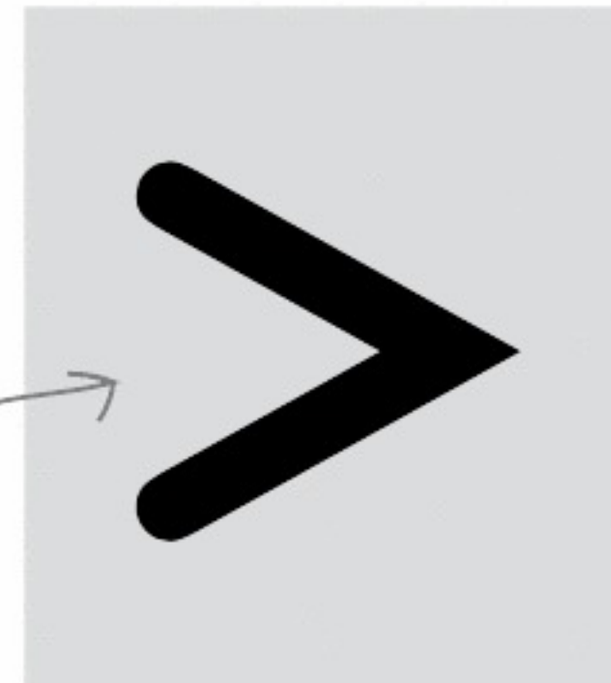
Le signe **inférieur** regarde les valeurs de la colonne à sa gauche et les compare à la valeur à sa droite. Si la valeur de la colonne est moins grande que celle de droite, cette ligne est renvoyée.

Le signe **supérieur** est le contraire d'inférieur. Il regarde les valeurs de la colonne et les compare à la valeur à sa droite. Si la valeur de la colonne est plus grande que la valeur à sa droite, cette ligne est renvoyée.



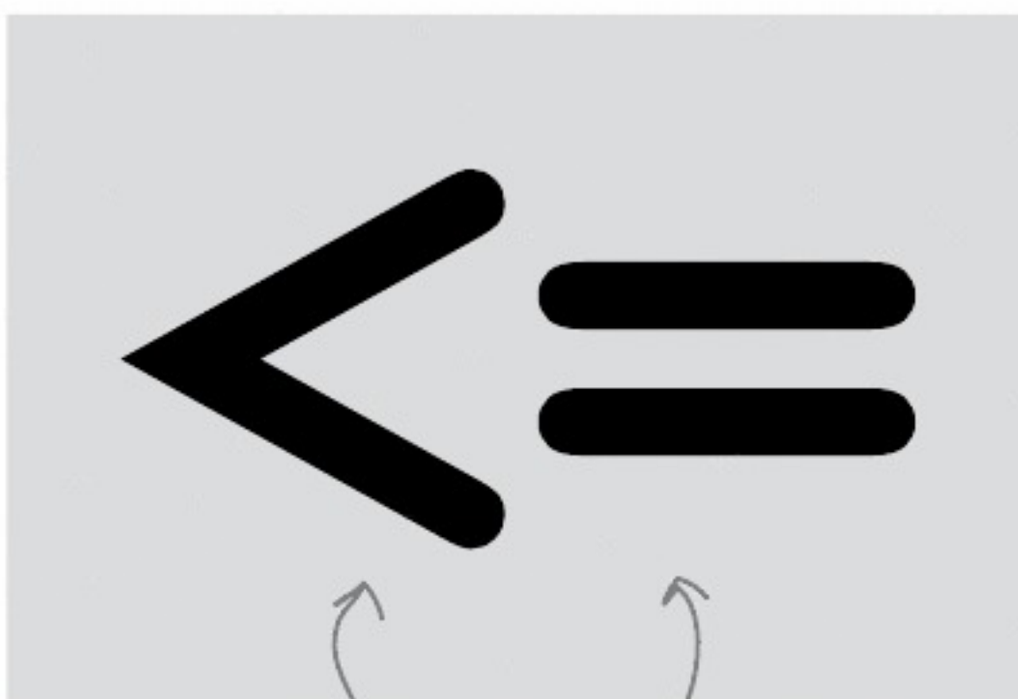
INFÉRIEUR renvoie toutes les valeurs moins grandes que la condition.

Et bien sûr il y a un SUPÉRIEUR.

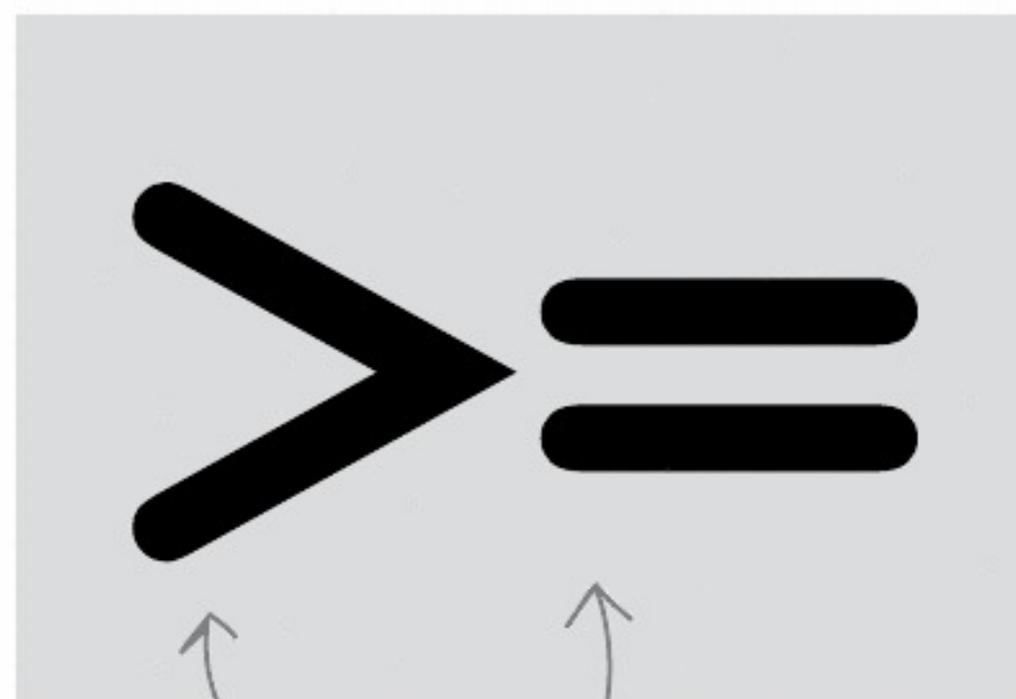


La seule différence avec le signe **inférieur ou égal à** est que les valeurs de la colonne égales à la valeur de la condition sont aussi renvoyées.

Même chose pour ce **signe supérieur ou égal à**. Si la valeur de la colonne correspond ou est plus grande que la valeur de la condition, la ligne est renvoyée.



Tous les enregistrements qui sont MOINS GRANDS QUE OU ÉGAUX À la condition sont renvoyés.



Et il y a un SUPÉRIEUR OU ÉGAL À.

Trouver des données numériques avec les opérateurs de comparaison

Le Bar Tête la première a une table avec les informations nutritives et le prix de leurs boissons. Ils veulent repérer des boissons les plus chères et les moins caloriques pour augmenter leurs bénéfices.

Ils utilisent des opérateurs de comparaison pour trouver les boissons qui sont vendues à au moins 3,50€ et ont moins de 50 calories dans la table `info_boissons`.

Le poids total de carbohydrates de chaque boisson en grammes.

Les calories de chaque boisson.

info_boissons

nom_boisson	prix	carbs	couleur	glace	calories
Épine noire	3	8.4	jaune	O	33
Lune bleue	2.5	3.2	bleu	O	12
Oh mon Dieu	3.5	8.6	orange	O	35
Tourbillon de citron vert	2.5	5.4	vert	O	24
Baiser sur les lèvres	5.5	42.5	violet	O	171
Or brûlant	3.2	32.1	orange	N	135
Arbre solitaire	3.6	4.2	rouge	O	17
Lévrier	4	14	jaune	O	50
Été indien	2.8	7.2	marron	N	30
Grenouille dingue	2.6	21.5	ébène	O	80
Soda et Ça	3.8	4.7	rouge	N	19

```
SELECT nom_boisson FROM info_boissons
WHERE
prix >= 3.5
AND
calories < 50;
```

Cela dit : « trouve les boissons qui coûtent 3,50 € ou plus. » Cela inclut les boissons à 3,50 €.

Cela dit : « trouve les boisson avec moins de 50 calories ».

Cette requête ne renvoie que les boissons où ces **deux** conditions sont remplies grâce au AND qui combine les deux résultats. Les boissons qui sont renvoyées sont : Oh mon Dieu, Arbre solitaire et Soda et Ça.

À vos crayons !

À vous de faire des mélanges. Écrivez des requêtes qui vont renvoyer les informations suivantes. Notez aussi le résultat de chaque requête :

Le prix de toutes les boissons jaunes avec de la glace et plus de 33 calories.

. .....
. .....
. .....
. .....
. .....
. .....

Résultat : .....

Le nom et la couleur de toutes les boissons qui ne contiennent pas plus de 4 grammes de carbohydrates et utilisent de la glace.

. .....
. .....
. .....
. .....
. .....
. .....

Résultat : . .....

Le prix de chaque boisson comptant 80 calories ou plus.

. .....
. .....
. .....
. .....
. .....
. .....
. .....

Résultat : . .....

Les boissons appelées Lévrier et Baiser sur les lèvres, ainsi que leur couleur et si on utilise de la glace pour mélanger la boisson.

. .....
. .....
. .....
. .....
. .....
. .....

Résultat : . .....

À vos crayons ! Solution

À vous de faire des mélanges. Écrivez des requêtes qui vont renvoyer les informations suivantes. Notez aussi le résultat de chaque requête :

Le prix de toutes les boissons jaunes avec de la glace et plus de 33 calories.

```
SELECT prix FROM info_boissons
WHERE glace = '0'
AND
couleur = 'jaune'
AND
calories > 33;
```

Résultat : 4.00 €

Le nom et la couleur de toutes les boissons qui ne contiennent pas plus de 4 grammes de carbohydrates et utilisent de la glace.

```
SELECT nom_boisson, couleur FROM info_boissons
WHERE
carbs <= 4
AND
glace = '0';
```

Résultat : Lune bleue, bleu

Le prix de chaque boisson comptant 80 calories ou plus.

```
SELECT prix FROM info_boissons
WHERE
calories >= 80;
```

Résultat : 5.50 €, 3.20 €, 2.60 €

Les boissons appelées Lévrier et Baiser sur les lèvres, ainsi que leur couleur et si on utilise de la glace pour mélanger la boisson.

```
SELECT nom_boisson, couleur, glace FROM info_boissons
WHERE
prix > 3.8;
```

Celui-ci est difficile. Vous deviez chercher dans la table une colonne que vous pouviez utiliser pour obtenir ces boissons et seulement celles-ci.

Résultat : Baiser sur les lèvres, violet, 0
Lévrier, jaune, 0

Trouver des données texte avec des opérateurs de comparaison

Comparer des données texte fonctionne de façon similaire dans vos colonnes de type CHAR et VARCHAR, par exemple. Les opérateurs de comparaison évaluent tout de façon **alphabétique**. Donc, si vous voulez sélectionner toutes les boissons qui commencent par un 'L', voici une requête qui va selectionner toutes les boissons qui répondent à ce critère.

info_boissons

nom_boisson	prix	carbs	couleur	glace	calories
Épine noire	3	8.4	jaune	O	33
Lune bleue	2.5	3.2	bleu	O	12
Oh mon Dieu	3.5	8.6	orange	O	35
Tourbillon de citron vert	2.5	5.4	vert	O	24
Baiser sur les lèvres	5.5	42.5	violet	O	171
Or brûlant	3.2	32.1	orange	N	135
Arbre solitaire	3.6	4.2	rouge	O	17
Lévrier	4	14	jaune	O	50
Été indien	2.8	7.2	marron	N	30
Grenouille dingue	2.6	21.5	ébène	O	80
Soda et Ça	3.8	4.7	rouge	N	19

```
SELECT nom_boisson
FROM info_boissons
WHERE
nom_boisson >= 'L'
AND
nom_boisson < 'M';
```

Cette requête renvoie les boissons dont la première lettre est L ou une lettre suivante mais dont la première lettre vient avant M dans l'alphabet.

Ne vous inquiétez pas de l'ordre de vos résultats pour l'instant.

Dans un prochain chapitre nous vous montrerons des façons de trier vos résultats dans l'ordre alphabétique.

Sélectionner vos ingrédients

On vient de demander à l'un des barmen de préparer un cocktail contenant du jus de cerise. Le barman peut utiliser deux requêtes pour trouver les cocktails :

Chaque requête vérifie les deux colonnes des ingrédients.

```
Fichier Edition Fenêtre Aide..
> SELECT nom_boisson FROM boissons_faciles WHERE principal = 'jus de
cerise';
+-----------------------+
| nom_boisson           |
+-----------------------+
| Baiser sur les lèvres |
+-----------------------+
1 row in set (0.02 sec)

> SELECT nom_boisson FROM boissons_faciles WHERE second = 'jus de
cerise';
+-----------------+
| nom_boisson     |
+-----------------+
| Arbre solitaire |
+-----------------+
1 row in set (0.01 sec)
```

Cela me semble vraiment inefficace. Il doit y avoir un moyen de combiner ces requêtes.

info_boissons

nom_boisson	prix	carbs	couleur	glace	calories
Épine noire	3	8.4	jaune	O	33
Lune bleue	2.5	3.2	bleu	O	12
Oh mon Dieu	3.5	8.6	orange	O	35
Tourbillon de citron vert	2.5	5.4	vert	O	24
Baiser sur les lèvres	5.5	42.5	violet	O	171
Or brûlant	3.2	32.1	orange	N	135
Arbre solitaire	3.6	4.2	rouge	O	17
Lévrier	4	14	jaune	O	50
Été indien	2.8	7.2	marron	N	30
Grenouille dingue	2.6	21.5	ébène	O	80
Soda et Ça	3.8	4.7	rouge	N	19

Être OU ne pas être

Vous pouvez combiner ces deux requêtes en utilisant `OR`. Cette condition renvoie des enregistrements quand ***l'une*** des conditions est remplie. Donc, au lieu de faire deux requêtes séparées, vous pouvez les combiner avec `OR`, de cette façon :

```
Fichier Edition Fenêtre Aide.. ChèreTarteAuxCerises
> SELECT nom_boisson from boissons_faciles
WHERE principal = 'jus de cerise'
OR
second = 'jus de cerise';
+-----------------------+
| nom_boisson           |
+-----------------------+
| Baiser sur les lèvres |
| Arbre solitaire       |
+-----------------------+
2 rows in set (0.02 sec)
```

Rayez les parties inutiles des deux instructions SELECT ci-dessous et ajoutez un OR pour en faire une seule instruction SELECT.

```
SELECT nom_boisson FROM boissons_faciles WHERE
principal = 'jus d\'orange';
```

```
SELECT nom_boisson FROM boissons_faciles WHERE
principal = 'jus de pomme';
```

Utilisez vos nouveaux talents de sélection pour réécrire un nouveau SELECT.

. .....

. .....

À vos crayons ! Solution

Rayez les parties inutiles des deux instructions SELECT ci-dessous et ajoutez un OR pour en faire une seule instruction SELECT.

```
SELECT nom_boisson FROM boissons_faciles WHERE
principal = 'jus d\'orange';
```

Nous devons nous débarrasser de ce point-virgule pour que l'instruction ne se termine pas à cet endroit.

OR

```
SELECT nom_boisson FROM boissons_faciles WHERE
principal = 'jus de pomme';
```

Avec ce OR nous obtenons les nom_boisson contenant comme ingrédient principal du jus d'orange OU du jus de pomme.

Nous pouvons simplement rayer cette ligne, nous avons déjà cela dans la première partie de la requête (reliée à présent par notre OR).

Utilisez vos nouveaux talents de sélection pour réécrire un nouveau SELECT.

```
SELECT nom_boisson FROM boissons_faciles
WHERE
principal = 'jus d\'orange'
OR
principal = 'jus de pomme';
```

Voici la requête finale.

Ne mélangez pas le AND et le OR !

Si vous voulez que **TOUTES** vos conditions soient vraies, utilisez **AND**.

Si vous voulez qu'**AU MOINS UNE** de vos conditions soit vraie, utilisez **OR**.

Toujours perdu ? Tournez la page.

AND

OR

il n'y a pas de

Questions stupides

Q: Peut-on utiliser plus d'un AND ou OR dans la même clause WHERE ?

R: Certainement. Vous pouvez en combiner autant que vous le désirez. Vous pouvez également utiliser AND et OR ensemble, dans la même clause.

La différence entre AND et OR

Dans les requêtes ci-dessous, vous allez voir des exemples de toutes les combinaisons possibles de deux conditions avec AND et OR entre elles.

evaluation_crepes

lieu	heure	date	type	note	commentaires
Le roi de la croustille	8:50	9/27	sucre	10	presque parfaite
Crêperie du port	8:59	8/25	NULL	6	grasse
Café Starbuzz	19:35	5/24	tarte au citron	5	tiède mais savoureuse
Crêperie du port	19:03	4/26	confiture	7	manque de confiture

```
SELECT type FROM evaluation_crepes                                  RÉSULTATS

WHERE lieu = 'Le roi de la croustille' AND note = 10;               sucre

WHERE lieu = 'Le roi de la croustille' OR note = 10;                sucre

WHERE lieu = 'Le roi de la croustille' AND note = 3;                pas de résultats

WHERE lieu = 'Le roi de la croustille' OR note = 3;                 sucre

WHERE lieu = 'Le bretzel dansant' AND note = 10;                    pas de résultats

WHERE lieu = 'Le bretzel dansant' OR note = 10;                     sucre

WHERE lieu = 'Le bretzel dansant' AND note = 3;                     pas de résultats

WHERE lieu = 'Le bretzel dansant' OR  note = 3;                     pas de résultats
```

Oui, il y a une correspondance

Oui

Pas de correspondance

Pas de correspondance

Vous ÊTES le Conditionnel

Vous trouverez ci-dessous une série de clauses WHERE avec des AND et des OR. Ne faites qu'un avec ces clauses et déterminez si elles vont, ou non, donner des résultats.

```
SELECT type FROM evaluation_crepes
```

Avez-vous obtenu un résultat ?

```
WHERE lieu  = 'Le roi de la croustille' AND  note <> 6;
```
.....

```
WHERE lieu  = 'Le roi de la croustille' AND  note = 3;
```
.....

```
WHERE lieu  = 'Le bretzel dansant' AND  note >= 6;
```
.....

```
WHERE lieu  = 'Le roi de la croustille' OR  note > 5;
```
.....

```
WHERE lieu  = 'Le roi de la croustille' OR  note = 3;
```
.....

```
WHERE lieu  = 'Le bretzel dansant' OR  note = 6;
```
.....

Pour améliorer votre karma, notez pourquoi deux de vos résultats sont un peu différents du reste.

Vous ÊTES le Conditionnel Solution

Vous trouverez ci-dessous une série de clauses WHERE avec des AND et des OR. Ne faites qu'un avec ces clauses et déterminez si elles vont, ou non, donner des résultats.

```
SELECT type FROM evaluation_crepes
```

Avez-vous obtenu un résultat ?

```
WHERE lieu  = 'Le roi de la croustille' AND  note <> 6;
```
sucre

```
WHERE lieu  = 'Le roi de la croustille' AND  note = 3;
```
pas de résultat

```
WHERE lieu  = 'Le bretzel dansant' AND  note >= 6;
```
pas de résultat

```
WHERE lieu  = 'Le roi de la croustille' OR  note > 5;
```
sucre, NULL, confiture

```
WHERE lieu  = 'Le roi de la croustille' OR  note = 3;
```
sucre

```
WHERE lieu  = 'Le bretzel dansant' OR  note = 6;
```
NULL

Pour améliorer votre karma, notez pourquoi deux de vos résultats sont un peu différents du reste.

Deux requêtes renvoient NULL.

Ces valeurs NULL peuvent vous poser des problèmes dans de futures requêtes. Il vaut mieux entrer une valeur quelconque que de laisser une valeur NULL dans une colonne ***parce que les NULL ne peuvent pas être sélectionnés directement dans une table.***

Utilisez IS NULL pour trouver les NULL

J'ai essayé de sélectionner les valeurs NULL directement mais cela n'a pas fonctionné. Comment puis-je trouver les NULL dans mes tables ?

info_boissons

nom_boisson	prix	carbs	couleur	glace	calories
Vacances	NULL	14	NULL	O	50
Souffle de dragon	2.9	7.2	marron	N	NULL

Vous ne pouvez pas sélectionner une valeur NULL directement.

```
SELECT nom_boisson FROM info_boissons
WHERE
calories = NULL ;
```

Ne fonctionnera pas parce que rien ne peut être égal à NULL. C'est une valeur indéfinie.

```
SELECT nom_boisson FROM info_boissons
WHERE
calories = 0;
```

Ceci ne fonctionnera pas parce que NULL n'est pas la même chose que zéro.

```
SELECT nom_boisson FROM info_boissons
WHERE
calories = 'NULL';
```

Et ceci ne fonctionnera pas non plus parce que NULL n'est pas une chaîne de caractères.

Mais vous pouvez la sélectionner en utilisant des mots-clés.

```
SELECT nom_boisson
FROM info_boissons
WHERE
calories IS NULL;
```

La seule façon de sélectionner une valeur NULL directement est d'utiliser les mots-clés IS NULL.

Les mots-clés ne sont pas des chaînes de caractères, ils n'ont donc pas de guillemets.

il n'y a pas de

Questions stupides

Q: Vous dites que l'on ne peut pas « sélectionner directement » NULL sans utiliser IS NULL. Est-ce que cela veut dire que l'on peut le sélectionner indirectement ?

R: Oui. Si vous vouliez obtenir la valeur de cette colonne, vous pourriez utiliser une clause WHERE sur l'une ou l'autre des autres colonnes. Par exemple, votre résultat sera NULL si vous utilisez cette requête :

```
SELECT calories FROM info_boissons
WHERE nom_boisson = 'Souffle de dragon';
```

Q: À quoi ressemblerait vraiment le résultat de cette requête ?

R: Il ressemblerait exactement à ceci :

```
+-----------+
| calories  |
+-----------+
| NULL      |
+-----------+
```

Pendant ce temps-là, chez Greg...

Greg essaie de trouver toutes les personnes des villes commençant par « Saint » dans sa table `mes_contacts`. Voici une partie de la requête sur laquelle il travaille :

```
SELECT * FROM mes_contacts
WHERE
ville = 'St-Malo'
OR
ville = 'Saint-Malo'
OR
ville = 'Saint-Nectaire'
OR
ville = 'Saint-Omer'
OR
ville = 'Saint-Tropez'
OR
ville = 'Saint-Trop'
OR
ville = 'Saint-Denis'
OR
ville = 'Saint-Nicolas'
OR
ville = 'Saint-Romain'
OR
ville = 'Saint-Étienne'
OR
Et la liste continue encore et
encore…
```

Il sait qu'il a entré St-Malo au moins de ces deux façons. Et qu'en est-il de la typo ?

Gagner du temps avec un seul mot-clé : LIKE

Il y a tout simplement trop de villes, de variations et de typos possibles. Utiliser tous ces `OR` va prendre des heures à Greg. Heureusement, il existe un mot-clé pour gagner du temps : `LIKE` qui, utilisé avec un joker, cherche des parties d'une chaîne de caractères et renvoie toutes les correspondances.

Greg peut utiliser `LIKE` comme ceci :

```
SELECT * FROM mes_contacts
WHERE ville LIKE 'Saint%';
```

Placez un signe pourcentage à l'intérieur des guillemets simples. Cela indique à votre logiciel que vous cherchez toutes les valeurs de la colonne ville qui commencent par Saint.

Faire appel au joker

`LIKE` fait équipe avec deux caractères jokers. Les jokers sont des suppléants pour les caractères qui sont en fait à cet endroit. Assez semblable au joker d'un jeu de carte, un joker est égal à n'importe quel caractère d'une chaîne.

Sortez le joker.

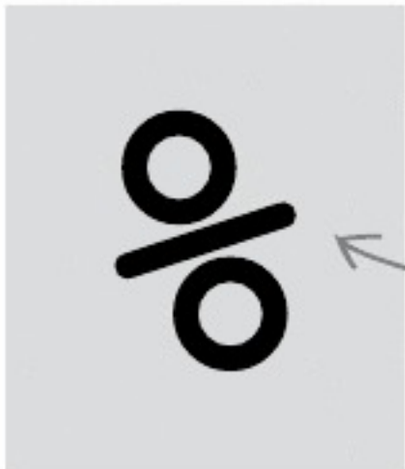
Les jokers sont des caractères suppléants.

GYM DU CERVEAU

Avez-vous vu d'autres jokers depuis le début de ce chapitre ?

C'est mieux COMME ça

LIKE aime jouer avec les jokers. Le premier est le signe de pourcentage, %, qui peut remplacer n'importe quel nombre de caractères inconnus.

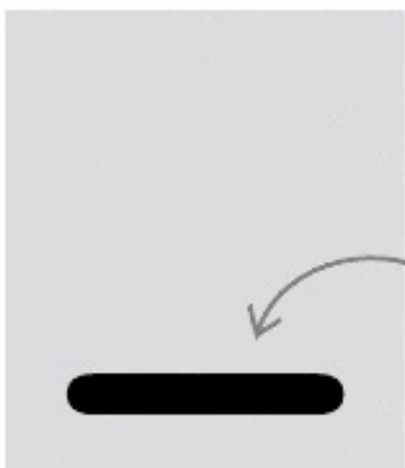


```
SELECT prenom FROM mes_contacts
WHERE prenom LIKE '%nne';
```

Le pourcentage est un remplaçant pour n'importe quel nombre de caractères inconnus.

Renvoie les noms avec n'importe quel nombre de caractères avant le « nne », comme Étienne, Marianne et Anne.

Le second caractère joker avec qui LIKE aime bien sortir est le tiret bas, _ qui remplace un seul caractère inconnu.



```
SELECT prenom FROM mes_contacts
WHERE prenom LIKE 'Ann_';
```

Le tiret bas est le remplaçant pour un seul caractère inconnu.

Renvoie des noms avec un seul caractère après le « Ann », comme Anne et Anna.

Le frigo

Un tas de clauses WHERE avec des LIKE sont mélangées sur le frigo. Pouvez-vous faire correspondre les clauses avec leurs résultats ? Certaines peuvent avoir plusieurs réponses. Écrivez vos propres instructions LIKE avec des jokers pour les résultats qui restent.

Joanna

Liberté

Mayenne

Splendide

Allier

Tendre

Elsie

Lozère

Morbihan

Mineur

Joshua

SQL Tête la première

Marne

Loiret

Splendeur

```
WHERE dpmt LIKE 'L%';
```

```
WHERE nom_vache LIKE '__sie';
```

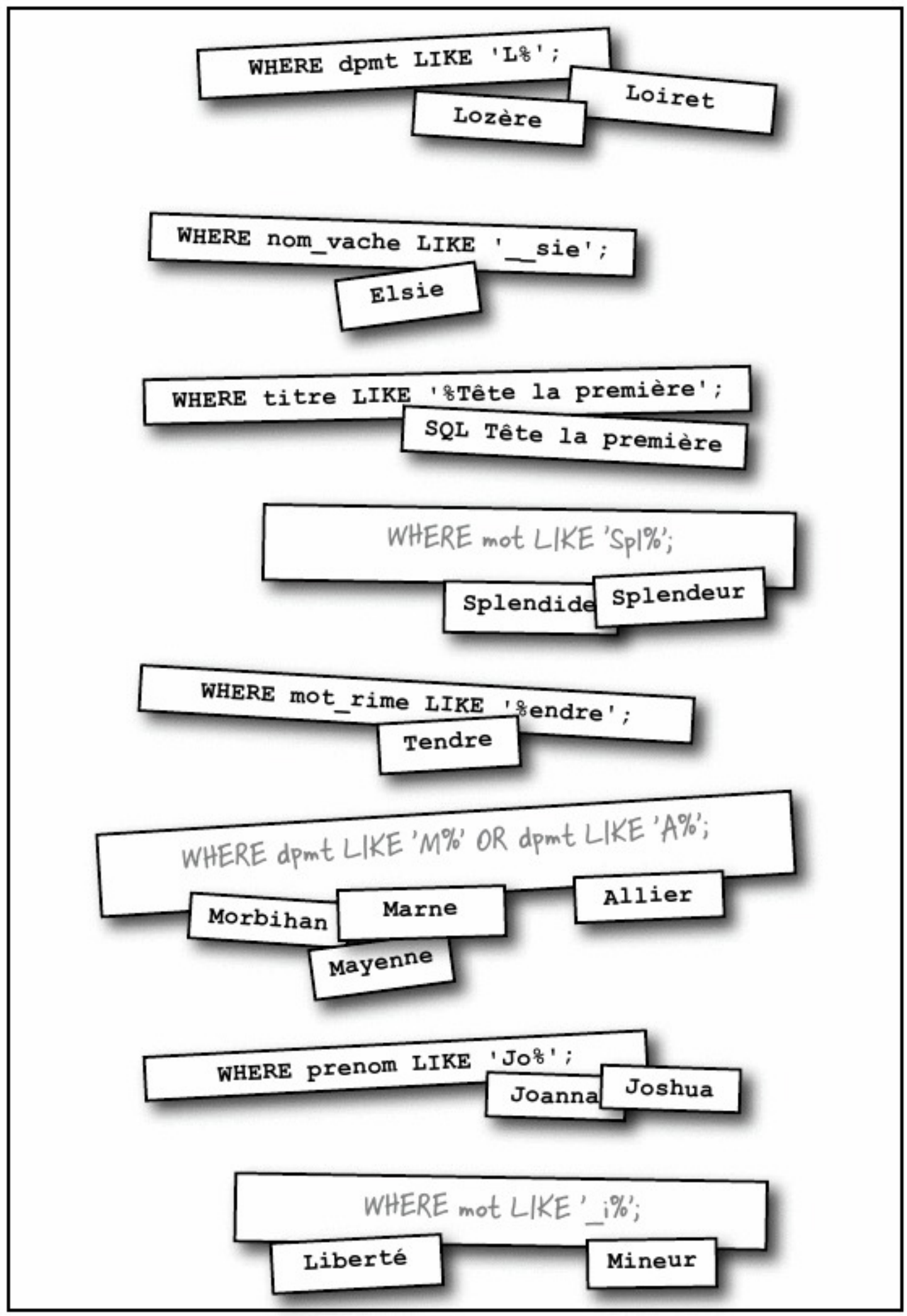
```
WHERE titre LIKE '%Tête la première';
```

```
WHERE mot_rime LIKE '%endre';
```

```
WHERE prenom LIKE 'Jo%';
```

Solutions du frigo

Un tas de clauses WHERE avec des LIKE sont mélangées sur le frigo. Pouvez-vous faire correspondre les clauses avec leurs résultats ? Certaines peuvent avoir plusieurs réponses. Écrivez vos propres instructions LIKE avec des jokers pour les résultats qui restent.



Sélectionner des rangées avec AND et les opérateurs de comparaison

Au Bar Tête la première, on essaie de repérer les boissons dans un certain ensemble de calories. Comment vont-ils formuler leur requête pour trouver les noms de boissons dont le nombre de calories est égal à ou compris entre 30 et 60 ?

info_boissons

nom_boisson	prix	carbs	couleur	glace	calories
Épine noire	3	8.4	jaune	O	33
Lune bleue	2.5	3.2	bleu	O	12
Oh mon Dieu	3.5	8.6	orange	O	35
Tourbillon de citron vert	2.5	5.4	vert	O	24
Baiser sur les lèvres	5.5	42.5	violet	O	171
Or brûlant	3.2	32.1	orange	N	135
Arbre solitaire	3.6	4.2	rouge	O	17
Lévrier	4	14	jaune	O	50
Été indien	2.8	7.2	marron	N	30
Grenouille dingue	2.6	21.5	ébène	O	80
Soda et Ça	3.8	4.7	rouge	N	19

```
SELECT nom_boisson FROM info_boissons
WHERE
calories >= 30
AND
calories <= 60;
```

Les résultats inclueront les boissons comptant 30 calories s'il y en a, ainsi que les boissons de 60 calories et les boissons dont le nombre de calories se situe entre les deux.

Juste ENTRE nous... on peut faire mieux

Nous pouvons utiliser le mot-clé `BETWEEN` à la place. Non seulement il raccourcit la requête mais, en plus, il donne les mêmes résultats. Remarquez que les limites (30 et 60) sont également incluses. `BETWEEN` est l'équivalent des symboles <= et >= , mais pas des symboles < et >.

```
SELECT nom_boisson FROM info_boissons
WHERE
calories BETWEEN 30 AND 60;
```

Cela vous donnera exactement les mêmes résultats que la requête de la page précédente mais regardez comme c'est plus rapide à taper !

Cela inclut les boissons de 30 et 60 calories.

Fichier Edition Fenêtre Aide CaloriesMoyennes

```
> SELECT nom_boisson FROM info_boissons
WHERE
calories BETWEEN 30 AND 60;
+------------------+
| nom_boisson      |
+------------------+
| Épine noire      |
| Oh mon Dieu      |
| Lévrier          |
| Été indien       |
+------------------+
```

À vos crayons !

Réécrivez la requête de la page précédente pour SELECTionner tous les noms de boissons qui ont plus de 60 calories et moins de 30.

. .....

. .....

. .....

Essayez d'utiliser BETWEEN sur des colonnes de texte. Écrivez une requête qui va SELECTionner les noms de boissons commençant par les lettres de G à O.

. .....

. .....

. .....

À votre avis, quel sera le résultat de cette requête ?

```
SELECT nom_boisson FROM info_boissons WHERE
calories BETWEEN 60 AND 30;
```

. .....

. .....

. .....

À vos crayons ! Solution

Réécrivez la requête de la page précédente pour SELECTionner tous les noms de boissons qui ont plus de 60 calories et moins de 30.

```
SELECT nom_boisson FROM info_boissons
WHERE
calories < 30 OR calories > 60;
```

Ceci nous donne les noms de boissons avec plus de 60 calories.

et ceci nous donne ceux de moins de 30 calories.

Essayez d'utiliser BETWEEN sur des colonnes de texte. Écrivez une requête qui va SELECTionner les noms de boissons commençant par les lettres de G à O.

```
SELECT nom_boisson FROM info_boissons
WHERE
nom_boisson BETWEEN 'G' AND 'O';
```

Nous obtiendrons les noms de boissons commençant par G et O, ainsi que par toutes les lettres entre les deux.

À votre avis, quel sera le résultat de cette requête ?

```
SELECT nom_boisson FROM info_boissons
WHERE
calories BETWEEN 60 AND 30;
```

L'ordre est important. Cette requête ne renverra donc aucun résultat.

Nous cherchons des valeurs entre 60 et 30. Il n'existe aucune valeur entre 60 et 30, parce que 60 vient après 30 dans l'ordre numérique. Le nombre le plus petit doit toujours être le premier pour que le **BETWEEN** soit interprété comme vous le voulez.

Après un rendez-vous, soit vous êtes IN...

Amandine, une amie de Greg, utilise les contacts de Greg pour rencontrer des hommes. Elle a eu un certain nombre de rendez-vous et a commencé à tenir une table « petit livre noir » avec ses impressions sur ses rendez-vous.

Elle a appelé sa table `livre_noir`. Elle veut obtenir une liste des bons rendez-vous, elle utilise donc ses notes positives.

```
SELECT nom_rdv
FROM livre_noir
WHERE
note = 'innovant'
OR
note = 'fabuleux'
OR
... ;
```

Il vous faut une ligne pour chaque appréciation positive.

Voici les appréciations positives.

livre_noir

nom_rdv	note
Alex	innovant
Julien	ennuyeux
Ian	fabuleux
Boris	hum hum
Martin	vulgaire
Éric	pathétique
Anthony	délicieux
Samuel	pas mal
Ivan	déséspéré
Vic	ridicule

Plutôt que d'utiliser tous ces OR, nous pouvons simplifier cela avec le mot-clé `IN`. Utilisez IN avec un ensemble de valeurs entre parenthèses. Quand une valeur de la colonne correspond à l'une des valeurs de l'ensemble, la ligne ou les colonnes spécifiées sont renvoyées.

```
SELECT nom_rdv
FROM livre_noir
WHERE
note IN ('innovant',
'fabuleux', 'délicieux',
'pas mal');
```

Utiliser le mot-clé IN signale à votre SGBDR qu'un ensemble de valeurs va arriver.

Voici l'ensemble des appréciations positives.

Fichier Édition Fenêtre Aide RDVSympas

```
> SELECT nom_rdv FROM livre_noir
WHERE

note IN ('innovant', 'fabuleux',
'délicieux', 'pas mal');


+------- ----+
| nom_rdv    |
+------------+
| Alex       |
| Ian        |
| Anthony    |
| Samuel     |
+------------+
```

... soit vous êtes NOT IN

Bien sûr, Amandine veut savoir qui a obtenu de mauvaises appréciations pour que, s'ils appellent, elle puisse dire qu'elle se lave les cheveux ou autre chose de très prenant.

Pour trouver les noms de ceux à qui elle a donné de mauvaises appréciations, nous allons ajouter le mot-clé `NOT` à notre instruction `IN`. `NOT` vous donne les résultats opposés, tout ce qui ne correspond pas à l'ensemble.

```
SELECT nom_rdv
FROM livre_noir
WHERE
note NOT IN ('innovant',
'fabuleux', 'délicieux',
'pas mal');
```

Utiliser les mots-clés NOT IN signale à votre logiciel que les résultats ne sont pas dans l'ensemble des termes.

Les résultats de la requête NOT IN sont les personnes qui n'ont pas obtenu d'appréciation positive et n'auront donc pas de deuxième rendez-vous.

```
Fichier Edition Fenêtre Aide RendezVousPourris
> SELECT nom_rdv FROM livre_noir
WHERE

note NOT IN ('innovant', 'fabuleux',
'délicieux', 'pas mal');

+------- ----+
| nom_rdv  |
+------------+
| Julien     |
| Boris      |
| Martin     |
| Éric       |
| Ivan       |
| Vic        |
+------------+
6 rows in set (2.43 sec)
```

MUSCLEZ VOS NEURONES

Pourquoi pourriez-vous choisir d'utiliser `NOT IN` plutôt qu'`IN` ?

Encore des NOT

Vous pouvez utiliser NOT avec BETWEEN et LIKE tout comme vous le pouvez avec IN. La chose à ne pas oublier est que **NOT vient juste après WHERE** dans votre instruction. Voici quelques exemples.

```
SELECT nom_boisson FROM info_boissons
WHERE NOT carbs BETWEEN 3 AND 5;
```

```
SELECT nom_rdv from livre_noir
WHERE NOT nom_rdv LIKE 'A%'
AND NOT nom_rdv LIKE 'B%';
```

Quand vous utilisez NOT avec AND ou OR, il vient juste après le AND ou le OR.

il n'y a pas de

Questions stupides

Q: Attendez, vous venez de dire que NOT va après WHERE. Mais quand on utilise NOT IN ?

R: C'est une exception. Et même si vous déplacez le NOT après le WHERE, cela fonctionnera. Ces deux instructions vous donneront exactement les mêmes résultats :

```
SELECT * FROM boissons_faciles
WHERE NOT principal IN ('soda', 'thé
glacé');
```

```
SELECT * FROM boissons_faciles
WHERE principal NOT IN ('soda', 'thé
glacé');
```

Q: Est-ce que cela fonctionnerait avec <>, l'opérateur de comparaison « non égal à » ?

R: Vous pourriez, mais c'est une double négation. Cela sera beaucoup plus logique d'utiliser un signe égal tout simplement. Ces deux requêtes renvoient les mêmes résultats :

```
SELECT * FROM boissons_faciles
WHERE NOT nom_boisson <> 'Épine noire';
```

```
SELECT * FROM boissons_faciles
WHERE nom_boisson = 'Épine noire';
```

Q: Comment cela fonctionnerait-il avec NULL ?

R: Exactement comme vous pouvez le deviner. Pour obtenir toutes les valeurs d'une colonne qui ne sont pas NULL, vous pouvez utiliser ceci :

```
SELECT * FROM boissons_faciles
WHERE NOT principal IS NULL;
```

Mais ceci fonctionnera aussi :

```
SELECT * FROM boissons_faciles
WHERE principal IS NOT NULL;
```

Q: Et avec AND et OR ?

R: Si vous voulez l'utiliser dans une clause AND ou OR, il va juste après ces mots, comme ceci :

```
SELECT * FROM boissons_faciles
WHERE NOT principal = 'soda'
AND NOT principal = 'thé glacé';
```

Réécrivez chacune des clauses WHERE suivantes pour qu'elles soient aussi simples que possible. Pour vous aider, vous pouvez utiliser AND, OR, NOT, BETWEEN, LIKE, IN, IS NULL et les opérateurs de comparaison. Utilisez les tables de ce chapitre comme références.

```
SELECT nom_boisson from boissons_faciles
WHERE NOT qte1 < 1.50;
```

.....

.....

.....

```
SELECT nom_boisson FROM info_boissons
WHERE NOT glace = 'O';
```

.....

.....

.....

```
SELECT nom_boisson FROM info_boissons
WHERE NOT calories < 20;
```

.....

.....

.....

```
SELECT nom_boisson FROM boissons_faciles
WHERE principal = 'nectar de pêche'
OR principal = 'soda' OR principal =
'Sprite' OR principal = 'Schweppes';
```

. .....

. .....

```
SELECT nom_boisson FROM info_boissons
WHERE NOT calories = 0;
```

. .....

. .....

. .....

```
SELECT nom_boisson FROM info_boissons
WHERE NOT carbs BETWEEN 3 AND 5;
```

. .....

. .....

. .....

```
SELECT nom_rdv from livre_noir
WHERE NOT nom_rdv LIKE 'A%'
AND NOT nom_rdv LIKE 'B%';
```

. .....

. .....

. .....

Réécrivez chacune des clauses WHERE suivantes pour qu'elles soient aussi simples que possible. Pour vous aider, vous pouvez utiliser AND, OR, NOT, BETWEEN, LIKE, IN, IS NULL et les opérateurs de comparaison. Utilisez les tables de ce chapitre comme références.

```
SELECT nom_boisson from boissons_faciles
WHERE NOT qte1 < 1.50;
```

```
SELECT nom_boisson FROM boissons_faciles
WHERE qte1 >= 1.50;
```

```
SELECT nom_boisson FROM info_boissons
WHERE NOT glace = 'O';
```

```
SELECT nom_boisson FROM info_boissons
WHERE glace = 'N';
```

```
SELECT nom_boisson FROM info_boissons
WHERE NOT calories < 20;
```

```
SELECT nom_boisson FROM info_boissons
WHERE calories >= 20;
```

```
SELECT nom_boisson FROM boissons_faciles
WHERE principal = 'nectar de pêche'
OR principal = 'soda' OR principal =
'Sprite' OR principal = 'Schweppes'
```

```
SELECT nom_boisson FROM boissons_faciles
WHERE principal BETWEEN 'N' AND 'S';
```

Ceci ne fonctionnera que parce que nous n'avons aucun autre des ingrédients principaux qui remplissent la condition. Si notre table contenait du sirop de grenadine, cela ne fonctionnerait pas.

```
SELECT nom_boisson FROM info_boissons
WHERE NOT calories = 0;
```

```
SELECT nom_boisson FROM info_boissons
WHERE calories > 0;
```

Nous n'avons pas de calories négatives, donc nous ne risquons rien avec le signe supérieur à.

```
SELECT nom_boisson FROM info_boissons
WHERE NOT carbs BETWEEN 3 AND 5;
```

```
SELECT nom_boisson FROM info_boissons
WHERE carbs < 3
OR
carbs > 5;
```

```
SELECT nom_rdv from livre_noir
WHERE NOT nom_rdv LIKE 'A%'
AND NOT nom_rdv LIKE 'B%';
```

```
SELECT nom_rdv FROM livre_noir
WHERE nom_rdv NOT BETWEEN 'A' AND 'B';
```

Boîte à outils SQL

Le chapitre 2 est derrière vous et vous avez maintenant ajouté les opérateurs dans votre boîte à outils. Pour une liste exhaustive des outils du livre, voyez l'Annexe C.

SELECT *

Utilisez ceci pour sélectionner toutes les colonnes d'une table.

**S'échapper avec ' et **

Échappez aux apostrophes de vos données texte en plaçant un guillemet simple supplémentaire ou un antislash devant lui.

= <> < > <= >=

Vous avez de nombreux opérateurs d'égalité ou d'inégalité à votre disposition. .

IS NULL

Utilisez ceci pour créer une condition qui traque cette nuisance de valeur NULL.

AND et OR

Avec AND et OR, vous pouvez combiner vos instructions conditionnelles dans vos clauses WHERE pour plus de précision.

NOT

NOT vous permet de rendre vos résultats négatifs et d'obtenir les valeurs opposées.

BETWEEN

Vous permet de sélectionner des ensembles de valeurs.

LIKE avec % et _

Utilisez LIKE avec les jokers pour chercher dans des parties de chaînes de caractères.

Vos nouveaux outils : les opérateurs !

Solutions des exercices de la page 59.

Greg veut créer une table de cocktails que les barmen puissent consulter pour chercher des recettes lors de ses soirées de speed-dating. En utilisant ce que vous avez appris dans le premier chapitre, créez la table sur cette page et insérez-y les données présentées.

Cette table fait partie d'une base de données appelée **boissons**. Elle contient la table **boissons_faciles** avec des recettes de boissons qui n'ont que deux ingrédients.

```
CREATE DATABASE boissons;

USE boissons;

CREATE TABLE boissons_faciles

(nom_boisson VARCHAR(30), principal VARCHAR(20), qte1 DEC(3,1),
second VARCHAR(20), qte2 DEC(4,2), instructions VARCHAR(250));
```

Il vaut mieux laisser quelques caractères en plus au cas où vous ayez besoin un jour d'entrer un nom plus long que les noms existants.

N'oubliez pas : les types de données numériques ne prennent pas de guillemets !

```
INSERT INTO boissons_faciles

VALUES

('Épine noire', 'Schweppes', 1.5, 'jus d\'ananas', 1, 'mélangez avec de la
glace pilée, filtrez dans un verre à cocktail avec un ruban d\'écorce de
citron'),
('Lune bleue', 'soda', 1.5, 'jus de cassis', .75, 'mélangez avec de la glace
pilée, filtrez dans un verre à cocktail avec un ruban d\'écorce de citron'),
('Oh mon Dieu', 'nectar de pêche', 1, 'jus d\'ananas', 1, 'mélangez avec de la
glace, filtrez dans un verre à liqueur'),
('Tourbillon de citron vert', 'Sprite', 1.5, 'jus de citron vert', .75,
'mélangez avec de la glace pilée, filtrez dans un verre à cocktail'),
('Baiser sur les lèvres', 'jus de cerise', 2, 'nectar d\'abricot', 7, 'servez
sur de la glace avec une paille'),
('Or brûlant', 'nectar de pêche', 3,' jus d\'orange', 6, 'versez le jus
d\'orange chaud dans une grande tasse et ajoutez le nectar de pêche'),
('Arbre solitaire', 'soda', 1.5, 'jus de cerise', .75, 'mélangez avec de la
glace pilée, filtrez dans un verre à cocktail'),
('Lévrier', 'soda', 1.5, 'jus de pamplemousse', 5, 'servez sur de la glace
pilée, mélangez bien'),
('Été indien', 'jus de pomme', 2, 'thé chaud', 6, 'versez le jus dans une
grande tasse et recouvrez avec le thé chaud'),
('Grenouille dingue', 'thé glacé', 1.5, 'limonade', 5, 'servez sur des glaçons
avec une rondelle de citron vert'),
('Soda et Ça', 'soda', 2, 'jus de raisin', 1, 'secouez dans un verre à cocktail,
pas de glace');
```

L'ensemble de valeurs de chaque boisson est entre parenthèses.

Et entre deux boissons, il y a une virgule.

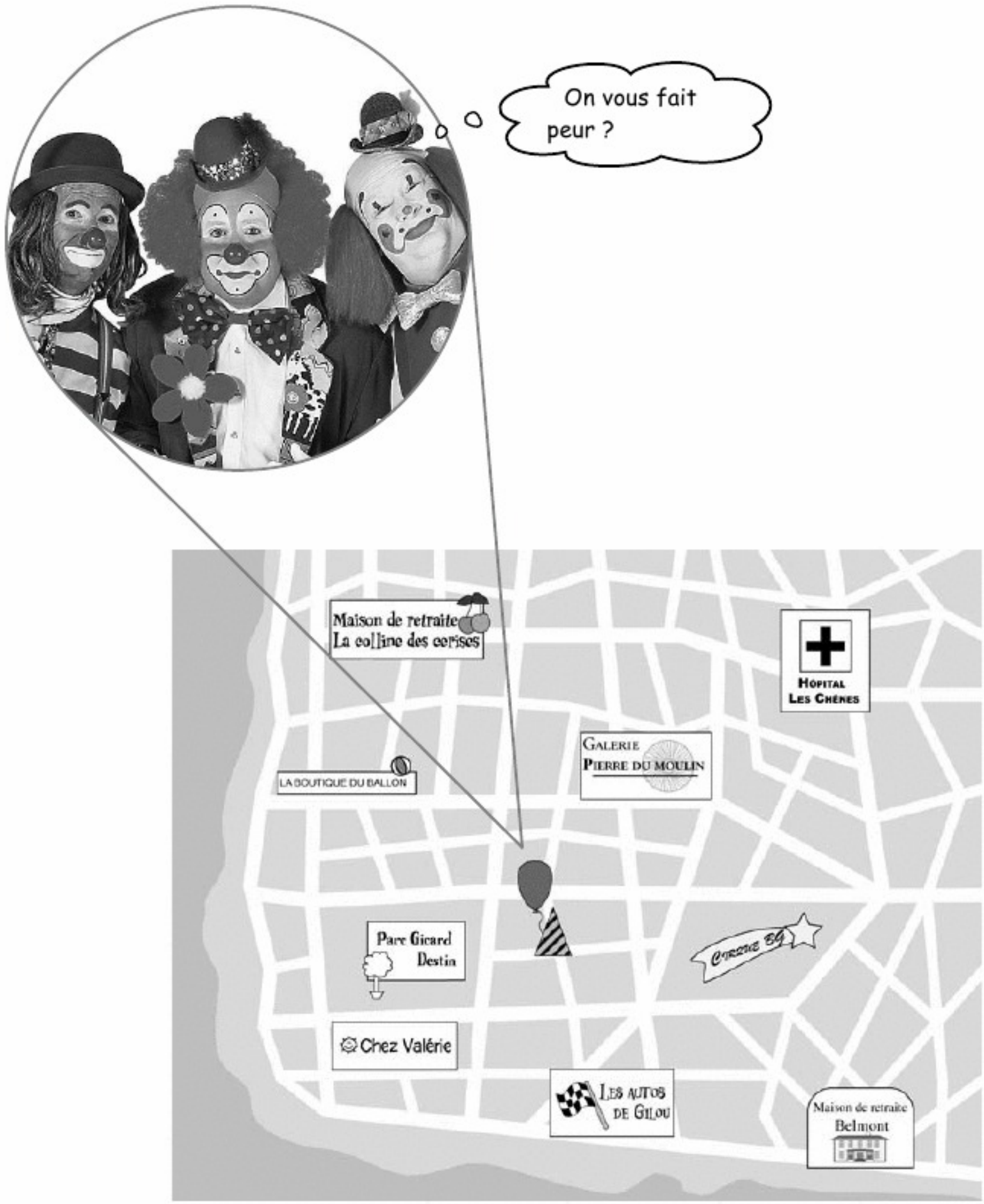
3 DELETE et UPDATE

Le changement vous fera du bien

Vous n'arrêtez pas de changer d'avis ? Maintenant vous pouvez ! Avec les commandes que vous êtes sur le point d'apprendre, **DELETE** et **UPDATE**, vous ne resterez plus coincé avec une décision prise six mois auparavant, quand vous avez commencé à insérer ces données sur les bananes qui, paraît-il, vont bientôt redevenir des coiffures à la mode. Avec UPDATE, vous **pouvez modifier les données** et DELETE vous permet de vous **débarrasser des données** dont vous n'avez plus besoin. Mais nous ne faisons pas que vous donner les outils ; dans ce chapitre, vous apprendrez à être sélectif avec vos nouveaux pouvoirs et à éviter de jeter des données dont vous avez réellement besoin.

Les clowns sont effrayants

Supposez que nous voulions répertorier les clowns de Dataville. Nous pouvons créer une table `info_clowns` pour les pister et nous pouvons utiliser une colonne `vu_en_dernier` pour garder une trace des trajets des clowns.



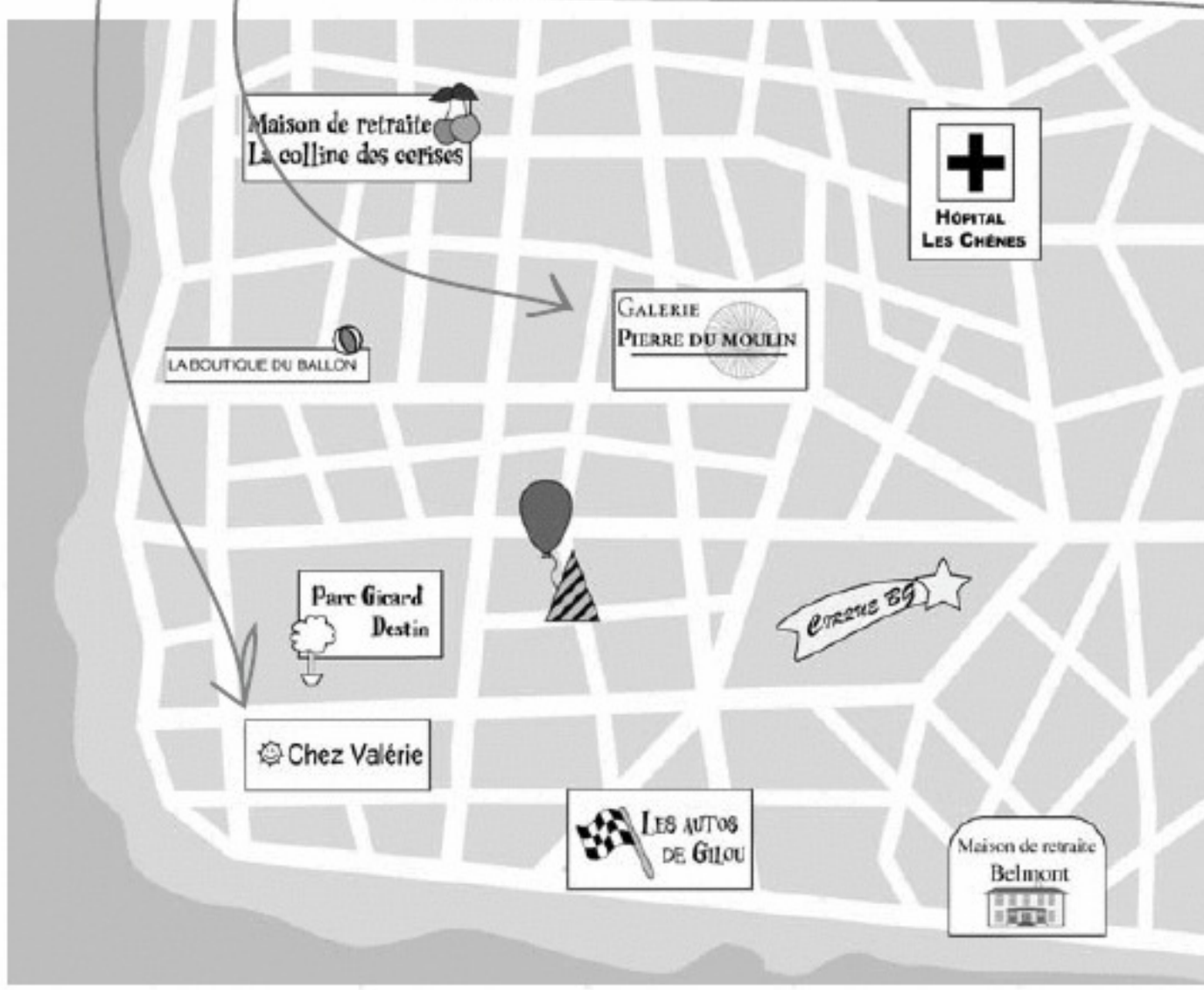
Sur la piste des clowns

Voici notre table. Nous pouvons laisser de côté les informations que nous ne connaissons pas et compléter la table plus tard. À chaque fois qu'un clown est vu quelque part, nous pouvons ajouter une nouvelle ligne. Nous allons devoir changer cette table souvent pour qu'elle reste à jour.

Là où chaque clown a été repéré en dernier.

info_clowns

nom	vu_en_dernier	description	activites
Elsie	Maison de retraite La colline des cerises	F, cheveux rouges, robe verte, pieds immenses	ballons, petite voiture
Cornichon	Fête de Max Leblanc	H, cheveux oranges, costume bleu, pieds immenses	mime
Mme Câlin	La boutique du ballon	F, chemise jaune, pantalon rouge trop grand	klaxon, parapluie
Bobo	Cirque BG	H, cigare, cheveux noirs, chapeau minuscule	violon
Clarabelle	Maison de retraite Belmont	F, cheveux roses, fleur énorme, robe bleue	cris, danse
Scooter	Hôpital Les Chênes	H, cheveux bleus, costume rouge, gros nez	ballons
Zippo	Galerie Pierre du moulin	F, costume orange, pantalon trop grand	danse
Poulette	Les autos de Gilou	F, tout en rose et strass	balançoire, petite voiture
Bozo		H, travesti, robe à pois	chant, danse
Sniff	Chez Valérie	H, costume vert et violet, nez pointu	



Nous compléterons les colonnes vides plus tard.

À vos crayons !

Les clowns ont bougé

Votre mission est d'écrire les commandes SQL pour saisir chaque rapport de terrain dans la table info_clowns. Remarquez que toutes les informations n'ont pas changé pour chaque clown, vous aurez donc besoin de vous référer à la table de la page 121 pour retrouver le reste des informations à ajouter.

Zippo vu en train de chanter

```
INSERT INTO info_clowns
VALUES
('Zippo', 'Galerie Pierre du moulin', 'F, costume orange,
pantalon trop grand', 'danse, chant');
```

Mme Câlin porte maintenant un pantalon bleu trop grand

```
INSERT INTO info_clowns
VALUES
('Mme Câlin', 'La boutique du ballon', 'F, chemise jaune,
pantalon bleu trop grand', 'klaxon, parapluie');
```

Bozo vu au Parc Gicard Destin

Sniff monte dans une petite voiture

Bobo vu récemment à la fête d'Éric Gris

Maintenant complétez la table info_clowns pour voir à quoi ressemblent les données une fois que vous avez tout ajouté en utilisant vos commandes INSERT.

nom	vu_en_dernier	description	activites
Elsie	Maison de retraite La colline des cerises	F, cheveux rouges, robe verte, pieds immenses	ballons, petite voiture
Cornichon	Fête de Max Leblanc	H, cheveux oranges, costume bleu, pieds immenses	mime
Mme Câlin	La boutique du ballon	F, chemise jaune, pantalon rouge trop grand	klaxon, parapluie
Bobo	Cirque BG	H, cigare, cheveux noirs, chapeau minuscule	violon
Clarabelle	Maison de retraite Belmont	F, cheveux roses, fleur énorme, robe bleue	cris, danse
Scooter	Hôpital Les Chênes	H, cheveux bleus, costume rouge, gros nez	ballons
Zippo	Galerie Pierre du moulin	F, costume orange, pantalon trop grand	danse
Poulette	Les autos de Gilou	F, tout en rose et strass	balançoire, petite voiture
Bozo		H, travesti, robe à pois	chant, danse
Sniff	Chez Valérie	H, costume vert et violet, nez pointu	

À vos crayons !

Les clowns ont bougé

Votre mission était d'écrire les commandes SQL pour saisir chaque rapport de terrain dans la table info_clowns, puis de compléter la table pour voir à quoi ressemblent ces données une fois que vous avez tout ajouté en utilisant vos commandes INSERT.

Zippo vu en train de chanter

```
INSERT INTO info_clowns
VALUES
('Zippo', 'Galerie Pierre du moulin', 'F, costume orange,
pantalon trop grand', 'danse, chant');
```

Mme Câlin porte maintenant un pantalon bleu trop grand

```
INSERT INTO info_clowns
VALUES
('Mme Câlin', 'La boutique du ballon', 'F, chemise jaune,
pantalon bleu trop grand', 'klaxon, parapluie');
```

Bozo vu au Parc Gicard Destin

```
INSERT INTO info_clowns
VALUES
('Bozo', 'Parc Gicard Destin', 'H, travesti, robe à pois',
'chant, danse');
```

Sniff monte dans une petite voiture

```
INSERT INTO info_clowns
VALUES
('Sniff', 'Chez Valérie', 'H, costume vert et violet, nez
pointu', 'monte dans une petite voiture');
```

Bobo vu récemment à la fête d'Éric Gris

```
INSERT INTO info_clowns
VALUES
('Bobo', 'Fête d\'Éric Gris', 'H, cigare, cheveux noirs
chapeau minuscule', 'violon');
```

nom	vu_en_dernier	description	activites
Elsie	Maison de retraite La colline des cerises	F, cheveux rouges, robe verte, pieds immenses	ballons, petite voiture
Cornichon	Fête de Max Leblanc	H, cheveux oranges, costume bleu, pieds immenses	mime
Mme Câlin	La boutique du ballon	F, chemise jaune, pantalon rouge trop grand	klaxon, parapluie
Bobo	Cirque BG	H, cigare, cheveux noirs, chapeau minuscule	violon
Clarabelle	Maison de retraite Belmont	F, cheveux roses, fleur énorme, robe bleue	cris, danse
Scooter	Hôpital Les Chênes	H, cheveux bleus, costume rouge, gros nez	ballons
Zippo	Galerie Pierre du moulin	F, costume orange, pantalon trop grand	danse
Poulette	Les autos de Gilou	F, tout en rose et strass	balançoire, petite voiture
Bozo		H, travesti, robe à pois	chant, danse
Sniff	Chez Valérie	H, costume vert et violet, nez pointu	
Zippo	Galerie Pierre du moulin	F, costume orange, pantalon trop grand	danse, chant
Mme Câlin	La boutique du ballon	F, chemise jaune, pantalon bleu trop grand	klaxon, parapluie
Bozo	Parc Gicard Destin	H, travesti, robe à pois	chant, danse
Sniff	Chez Valérie	H, costume vert et violet, nez pointu	monte dans une petite voiture
Bobo	Fête d'Éric Gris	H, cigare, cheveux noirs, chapeau minuscule	violon

MUSCLEZ VOS NEURONES

Comment pouvez-vous localiser un clown en particulier ?

Comment nos données sur les clowns sont-elles saisies ?

Nos espions de clowns travaillent bénévolement. Il arrive que les rapports d'espionnage de clowns attendent dans un casier pendant une semaine ou deux avant d'être saisis et parfois deux personnes **se partagent la pile** de rapports et **saisissent les données en même temps.**

En gardant cela à l'esprit, examinons toutes les lignes de notre table concernant Zippo. Nous pouvons lancer une instruction `SELECT` pour les obtenir :

```
Fichier Edition Fenêtre Aide AttrapeLeClown
SELECT * FROM info_clowns WHERE nom = 'Zippo';
```

nom	vu_en_dernier	description	activites
Zippo	Galerie Pierre du moulin	F, costume orange, pantalon trop grand	danse, chant
Zippo	Galerie Pierre du moulin	F, costume orange, pantalon trop grand	danse
Zippo	Hôpital Les Chênes	F, costume orange, pantalon trop grand	danse
Zippo	Chez Valérie	F, costume orange, pantalon trop grand	danse
Zippo	La boutique du ballon	F, costume orange, pantalon trop grand	danse
Zippo	Galerie Pierre du moulin	F, costume orange, pantalon trop grand	danse
Zippo	Hôpital Les Chênes	F, costume orange, pantalon trop grand	danse

Ces deux enregistrements sont exactement identiques.

Ceux-ci sont aussi identiques.

Cette info se répète encore et encore.

Existe-t-il un moyen de ne récupérer que nos données avec le dernier endroit où l'on a vu Zippo ? Pouvez-vous trouver cet endroit ?

Malheureusement, vous ne pouvez pas être certain que le dernier enregistrement est le plus récent.

Il y a plus d'une personne qui saisissent les données en même temps et les rapports peuvent avoir été mélangés dans le casier. Mais même si ce n'était pas le cas, **vous ne pouvez pas compter sur le fait que les lignes de la table soient dans l'ordre chronologique**.

De nombreux facteurs internes de la base de données peuvent changer l'ordre dans lequel les lignes d'une table sont enregistrées. Parmi ces facteurs, on trouve le SGBDR que vous utilisez et les index de vos colonnes (auxquels nous reviendrons plus tard).

Vous ne pouvez pas être sûr que la dernière ligne d'une table est la plus récemment ajoutée à cette table.

Bozo, nous avons un problème

Puisque l'on ne peut pas considérer que le dernier enregistrement est le plus récent, nous avons un problème. Notre table de clowns nous donne une liste des endroits où se trouvaient les clowns à un moment. **Mais la principale raison d'être de la table est de nous dire où les clowns ont été vus en dernier.**

Ce n'est pas tout. Vous voyez les enregistrements dupliqués ? Nous avons deux lignes montrant Zippo au même endroit et faisant la même chose. Elles prennent de la place et vont ralentir votre SGBDR au fur et à mesure que vos tables vont s'agrandir. Les enregistrements dupliqués **ne doivent pas exister dans une table**. Dans quelques chapitres, nous allons expliquer en quoi les duplicats sont mauvais et comment les éviter avec une ***bonne conception de table***. Vous verrez comment créer des tables qui n'auront jamais d'enregistrements dupliqués. Mais pour l'instant, concentrons-nous sur la façon de modifier notre table pour qu'elle contienne des données utiles.

il n'y a pas de Questions stupides

Q: Pourquoi ne peut-on pas simplement considérer que le dernier enregistrement est le plus récent ?

R: L'ordre des enregistrements dans une table n'est pas garanti et vous allez bientôt modifier l'ordre des résultats que vous obtenez. Vous ne pouvez pas avoir la certitude absolue que la dernière entrée est *vraiment* le dernier enregistrement inséré. De plus, une simple erreur humaine peut déranger l'ordre d'une table. Supposez que nous saisissions deux instructions `INSERT` pour le même clown. À moins de mettre un point d'honneur à se rappeler du rapport qui est arrivé en dernier, après que les données sont dans la table, nous ne savons pas avec certitude quel est le dernier rapport.

Q: Supposez que nous nous rappelions de l'ordre. Alors, pourquoi ne peut-on pas simplement utiliser le dernier enregistrement ?

R: Élargissons l'exemple. Nous avons suivi les mêmes clowns pendant de nombreuses années. Nous avons peut-être des assistants qui les suivent eux aussi et qui insèrent leurs propres enregistrements. Certains des clowns ont des centaines d'enregistrements. En utilisant SELECT, nous obtenons ces centaines d'enregistrements et nous devons nous frayer un chemin à travers eux jusqu'au dernier, qui, nous l'*espérons*, est le plus récent.

Q: Est-ce que cela arrive vraiment de vouloir garder des données comme cela dans une table ? Est-ce que cela a un sens d'insérer de nouveaux enregistrements et de garder les vieux ?

R: Absolument. Prenez notre exemple du moment. La table telle qu'elle est maintenant ne nous donne que le dernier endroit où un clown a été vu mais elle nous donne aussi un historique de leurs déplacements. Ce sont des informations potentiellement utiles. Le problème est que nous n'avons aucune information claire dans chaque enregistrement pour nous dire quand cela s'est produit. Si nous avions une colonne avec la date et l'heure, alors nous pourrions suivre les clowns avec une grande précision.

Mais pour l'instant, nous devons enlever ces enregistrements identiques de notre table pour simplifier les choses.

Q: D'accord, alors à la fin de ce livre je saurai comment concevoir des tables sans lignes dupliquées. Mais si la personne que je remplace m'a laissé une table mal conçue ?

R: Les tables mal conçues sont fréquentes dans le monde réel et la plupart des gens qui apprennent SQL se retrouvent à devoir arranger les bourdes SQL d'autres personnes.

Il existe un certain nombre de techniques pour nettoyer les lignes dupliquées. Certaines des meilleures utilisent des jointures, un sujet abordé plus loin dans ce livre. Pour l'instant vous n'avez pas tous les outils dont vous avez besoin pour vous occuper des mauvaises données, mais vous les aurez à la fin du livre.

Se débarrasser d'un enregistrement avec DELETE

On dirait que nous allons devoir nous débarrasser de quelques enregistrements. Pour que notre table nous soit plus utile, nous ne devons avoir qu'une ligne par clown. En attendant qu'arrive un nouveau rapport sur Zippo, dont nous saurons qu'il sera le plus récent, nous pouvons nous débarrasser de certains des vieux enregistrements de Zippo qui ne nous aident pas.

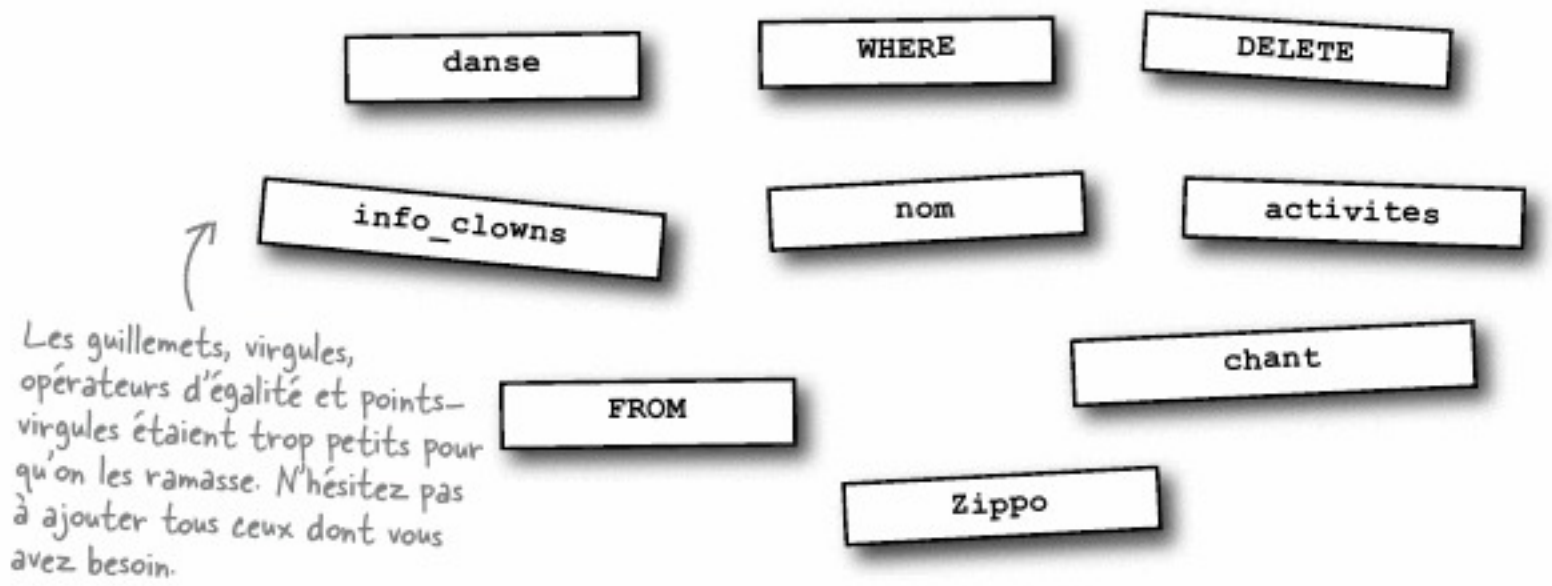
Votre outil pour supprimer les lignes de données de votre table est l'instruction `DELETE`. Elle utilise le même genre de clauses `WHERE` que vous connaissez déjà. Voyez si vous pouvez trouver la bonne syntaxe avant que nous ne vous la montrions.

Revoici les lignes pour Zippo :

nom	vu_en_dernier	description	activites
Zippo	Galerie Pierre du moulin	F, costume orange, pantalon trop grand	danse, chant
Zippo	Galerie Pierre du moulin	F, costume orange, pantalon trop grand	danse
Zippo	Hôpital Les Chênes	F, costume orange, pantalon trop grand	danse
Zippo	Chez Valérie	F, costume orange, pantalon trop grand	danse
Zippo	La boutique du ballon	F, costume orange, pantalon trop grand	danse
Zippo	Galerie Pierre du moulin	F, costume orange, pantalon trop grand	danse
Zippo	Hôpital Les Chênes	F, costume orange, pantalon trop grand	danse

DELETE sur le frigo

Nous avons écrit une commande simple que nous pouvons utiliser pour nous débarrasser de tous les enregistrements de Zippo sauf le premier, mais tous les morceaux sont tombés du frigo. Réassemblez les fragments et annotez-les en écrivant ce que fait, à votre avis, chaque partie de la nouvelle commande.



DELETE sur le frigo - Solution

Nous avons écrit une commande simple que nous pouvons utiliser pour nous débarrasser de tous les enregistrements de Zippo sauf le premier, mais tous les morceaux sont tombés du frigo. Réassemblez les fragments et annotez-les en écrivant ce que fait, à votre avis, chaque partie de la nouvelle commande.

Contrairement à l'instruction SELECT, nous n'avons pas besoin de lui dire quoi supprimer. Elle supprimera l'enregistrement en entier.

```
DELETE
FROM info_clowns
WHERE activites = 'danse';
```

Précise la table où il faut enlever la ligne.

Vous devez avoir ajouté un opérateur d'égalité, des guillemets autour de l'aimant danse et un point-virgule pour terminer la commande.

Faites attention de ne pas oublier votre clause WHERE, sinon vous allez supprimer toutes vos lignes !

Notre clause WHERE du chapitre précédent est vraiment très pratique ici. C'est comme cela que nous précisons exactement quel enregistrement on supprime avec DELETE.

Vous n'aviez pas besoin de ces aimants pour la commande.

```
chant
nom
zippo
```

Vous pouvez utiliser des clauses WHERE avec les instructions DELETE de la même façon que vous les utilisez avec les instructions SELECT.

Utiliser notre nouvelle instruction DELETE

Utilisons l'instruction DELETE que nous venons de créer. Elle fait exactement ce par quoi elle se traduit. Tous les enregistrements qui correspondent à la condition WHERE vont être supprimés de notre table.

```
DELETE FROM info_clowns
WHERE
activites = 'danse';
```

nom	vu_en_dernier	description	activites
Elsie	Maison de retraite La colline des cerises	F, cheveux rouges, robe verte, pieds immenses	ballons, petite voiture
Cornichon	Fête de Max Leblanc	H, cheveux oranges, costume bleu, pieds immenses	mime
Mme Câlin	La boutique du ballon	F, chemise jaune, pantalon rouge trop grand	klaxon, parapluie
Bobo	Cirque BG	H, cigare, cheveux noirs, chapeau minuscule	violon
Clarabelle	Maison de retraite Belmont	F, cheveux roses, fleur énorme, robe bleue	cris, danse
Scooter	Hôpital Les Chênes	H, cheveux bleus, costume rouge, gros nez	ballons
Zippo	Galerie Pierre du moulin	F, costume orange, pantalon trop grand	danse
Poulette	Les autos de Gilou	F, tout en rose et strass	balançoire, petite voiture
Bozo		H, travesti, robe à pois	chant, danse
Sniff	Chez Valérie	H, costume vert et violet, nez pointu	
Zippo	Galerie Pierre du moulin	F, costume orange, pantalon trop grand	danse, chant
Mme Câlin	La boutique du ballon	F, chemise jaune, pantalon bleu trop grand	klaxon, parapluie
Bozo	Parc Gicard Destin	H, travesti, robe à pois	chant, danse
Sniff	Chez Valérie	H, costume vert et violet, nez pointu	monte dans une petite voiture
Bobo	Fête pour Éric Gris	H, cigare, cheveux noirs, chapeau minuscule	violon

Voici l'enregistrement qui va être supprimé.

MUSCLEZ VOS NEURONES

Pensez-vous que l'on peut supprimer une seule colonne d'une ligne en utilisant DELETE ?

Les règles du DELETE

- Vous ne pouvez pas utiliser `DELETE` pour supprimer les valeurs d'une colonne ou d'une série de colonnes.
- Vous pouvez utiliser `DELETE` pour supprimer une ou plusieurs lignes, selon votre clause WHERE.
- Vous avez vu comment supprimer une ligne d'une table. Nous pouvons également supprimer plusieurs lignes d'une table. Pour cela, nous utilisons une clause `WHERE` pour indiquer à notre `DELETE` quelles lignes choisir. Cette clause `WHERE` est exactement la même que celle que vous avez utilisée dans le chapitre 2 avec vos instructions `SELECT`. Elle peut utiliser tout ce que vous avez vu dans le chapitre 2, comme `LIKE`, `IN`, `BETWEEN` et toutes les conditions pour indiquer précisément à votre SGBDR les lignes à supprimer.
- Enfin, faites attention à ceci, vous pouvez supprimer toutes les lignes d'une table avec :

```
DELETE FROM votre_table;
```

il n'y a pas de

Questions stupides

Q: Y a-t-il une différence entre utiliser un WHERE avec un DELETE et utiliser un WHERE avec SELECT ?

R: Aucune. Le WHERE est le même, mais ce que font SELECT et DELETE est complètement différent. SELECT renvoie une copie des colonnes des lignes qui correspondent à la condition WHERE, mais ne change pas votre table. DELETE enlève toutes les lignes qui correspondent à la condition WHERE. Il enlève une ligne toute entière de la table.

Reliez chaque requête à la ligne ou aux lignes qu'elle supprime :

```
DELETE FROM evaluation_crepes

WHERE lieu  = 'Le roi de la croustille' AND  note <> 6;

WHERE lieu  = 'Le roi de la croustille' AND  note = 3;

WHERE lieu  = 'Le bretzel dansant' AND  note >= 6;

WHERE lieu  = 'Le roi de la croustille' OR  note > 5;

WHERE lieu  = 'Le roi de la croustille' OR  note = 3;

WHERE lieu  = 'Le bretzel dansant' OR  note = 6;
```

evaluation_crepes

lieu	heure	date	type	note	commentaires
Le roi de la croustille	8:50	9/27	sucre glace	10	presque parfaite
La crêperie du port	8:59	8/25	NULL	NULL	grasse
Café Starbuzz	19:35	5/24	tarte au citron	5	pas frais mais savoureux
La crêperie du port	19:03	4/26	confiture	7	manque de confiture

Vous ÊTES le DELETE avec des clauses WHERE - Solution

Vous deviez incarner une série de DELETE comportant des clauses WHERE avec des AND et des OR pour déterminer s'ils allaient détruire des lignes ou pas.

Reliez chaque requête à la ligne ou aux lignes qu'elle supprime :

```
DELETE FROM evaluation_crepes
WHERE lieu  = 'Le roi de la croustille' AND  note <> 6;

WHERE lieu  = 'Le roi de la croustille' AND  note = 3;

WHERE lieu  = 'Le bretzel dansant' AND  note >= 6;

WHERE lieu  = 'Le roi de la croustille' OR  note > 5;

WHERE lieu  = 'Le roi de la croustille' OR  note = 3;

WHERE lieu  = 'Le bretzel dansant' OR  note = 6;
```

Pas de correspondances, ne supprime rien

Pas de correspondances, ne supprime rien

Pas de correspondances, ne supprime rien

evaluation_crepes

lieu	heure	date	type	note	commentaires
Le roi de la croustille	8:50	9/27	sucre glace	10	presque parfaite
La crêperie du port	8:59	8/25	NULL	NULL	grasse
Café Starbuzz	19:35	5/24	tarte au citron	5	pas frais mais savoureux
La crêperie du port	19:03	4/26	confiture	7	manque de confiture

Ces valeurs NULL peuvent vous poser des problèmes pour vos prochaines requêtes. Il vaut mieux saisir une valeur quelconque que de laisser une valeur NULL dans une colonne parce que les NULL ne peuvent pas être trouvés avec une condition d'égalité.

Les deux étapes de l'INSERT-DELETE

Il n'y a qu'un enregistrement pour Clarabelle dans toute la table. Comme nous ne voulons, pour chaque clown, que la ligne contenant leurs plus récentes informations, nous devons simplement créer un nouvel enregistrement et supprimer l'ancien.

Seule l'activité diffère de la ligne actuelle.

Clarabelle vue en train de danser à la Maison de retraite Belmont. F, cheveux roses, fleur énorme, robe bleue

Notre mission était d'ajouter ces données à cette table. Nous ne montrons qu'une ligne de la table de la page 131 pour des raisons d'espace.

nom	vu_en_dernier	description	activites
Clarabelle	Maison de retraite Belmont	F, cheveux roses, fleur énorme, robe bleue	cris, danse

1 D'abord, utilisez `INSERT` pour ajouter les nouvelles informations (et toutes les anciennes infos aussi).

```
INSERT INTO info_clowns
VALUES
('Clarabelle', 'Maison de retraite Belmont', 'F,
cheveux roses, fleur énorme, robe bleue', 'danse');
```

Insérez l'enregistrement en utilisant les données d'origine et ne modifiez que la colonne que vous avez besoin de changer.

nom	vu_en_dernier	description	activites
Clarabelle	Maison de retraite Belmont	F, cheveux roses, fleur énorme, robe bleue	cris, danse
Clarabelle	Maison de retraite Belmont	F, cheveux roses, fleur énorme, robe bleue	danse

INSERT

2 Puis, utilisez `DELETE` pour supprimer l'ancien enregistrement en utilisant une clause `WHERE`.

```
DELETE FROM info_clowns
WHERE
activites = 'cris, danse'
AND nom = 'Clarabelle';
```

Utilisez une clause WHERE pour trouver et supprimer l'ancien enregistrement.

Maintenant il ne nous reste que le nouvel enregistrement.

nom	vu_en_dernier	description	activites
Clarabelle	Maison de retraite Belmont	F, cheveux roses, fleur énorme, robe bleue	danse

À vos crayons !

Utilisez INSERT et DELETE pour changer la table info_boissons comme demandé, puis dessinez la table modifiée à droite.

info_boissons

nom_boisson	prix	carbs	couleur	glace	calories
Épine noire	3	8.4	jaune	O	33
Lune bleue	2.5	3.2	bleu	O	12
Oh mon Dieu	3.5	8.6	orange	O	35
Tourbillon de citron vert	2.5	5.4	vert	O	24
Baiser sur les lèvres	5.5	42.5	violet	O	171
Or brûlant	3.2	32.1	orange	N	135
Arbre solitaire	3.6	4.2	rouge	O	17
Lévrier	4	14	jaune	O	50
Été indien	2.8	7.2	marron	N	30
Grenouille dingue	2.6	21.5	ébène	O	80
Soda et Ça	3.8	4.7	rouge	N	19

Changez le nombre de calories de Baiser sur les lèvres en 170.

. .....

. .....

. .....

. .....

Changez les valeurs jaune en or.

. .....

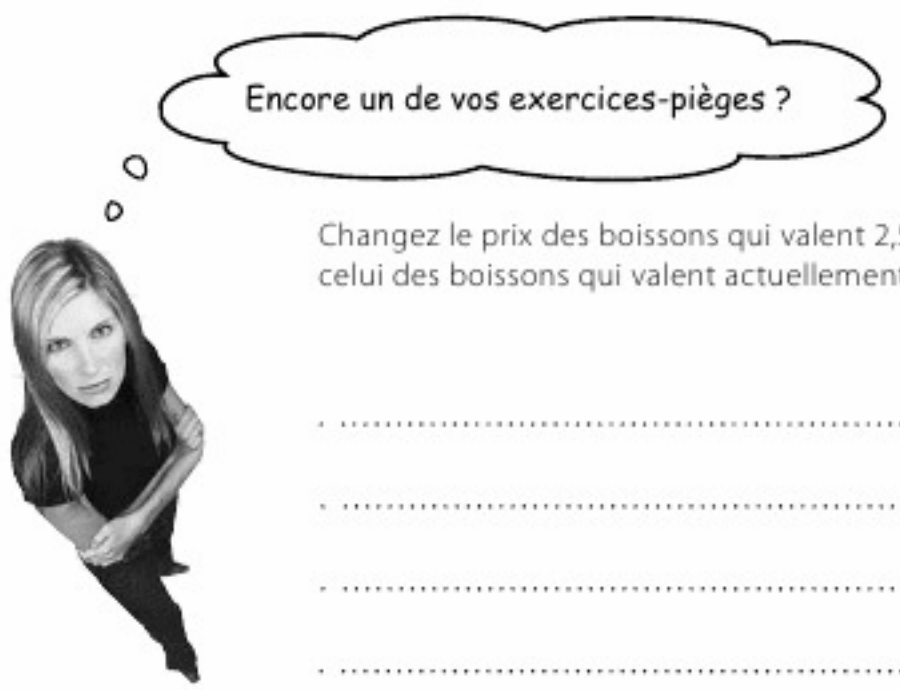
. .....

. .....

. .....

info_boissons

nom_boisson	prix	carbs	couleur	glace	calories
Épine noire					
Lune bleue					
Oh mon Dieu					
Tourbillon de citron vert					
Baiser sur les lèvres					
Or brûlant					
Arbre solitaire					
Lévrier					
Été indien					
Grenouille dingue					
Soda et Ça					



Changez le prix des boissons qui valent 2,50 € en 3,50 € et celui des boissons qui valent actuellement 3,50 € en 4,50 €.

. .....

. .....

. .....

. .....

. .....

À vos crayons ! Solution

Utilisez INSERT et DELETE pour changer la table info_boissons comme demandé, puis dessinez la table modifiée à droite.

info_boissons

nom_boisson	prix	carbs	couleur	glace	calories
Épine noire	3	8.4	jaune	O	33
Lune bleue	2.5	3.2	bleu	O	12
Oh mon Dieu	3.5	8.6	orange	O	35
Tourbillon de citron vert	2.5	5.4	vert	O	24
Baiser sur les lèvres	5.5	42.5	violet	O	171
Or brûlant	3.2	32.1	orange	N	135
Arbre solitaire	3.6	4.2	rouge	O	17
Lévrier	4	14	jaune	O	50
Été indien	2.8	7.2	marron	N	30
Grenouille dingue	2.6	21.5	ébène	O	80
Soda et Ça	3.8	4.7	rouge	N	19

Changez le nombre de calories de Baiser sur les lèvres en 170.

```
INSERT INTO info_boissons VALUES ('Baiser sur les lèvres', 5.5, 42.5, 'violet', '0', 170);
DELETE FROM info_boissons WHERE calories = 171;
```

Changez les valeurs jaune en or.

```
INSERT INTO info_boissons VALUES ('Epine noire', 3, 8.4, 'or', '0', 33),
('Lévrier', 4, 14, 'or', '0', 50);
DELETE FROM info_boissons WHERE couleur = 'jaune';
```

info_boissons

nom_boisson	prix	carbs	couleur	glace	calories
Épine noire	3	8.4	or	O	33
Lune bleue	3.5	3.2	bleu	O	12
Oh mon Dieu	4.5	8.6	orange	O	35
Tourbillon de citron vert	3.5	5.4	vert	O	24
Baiser sur les lèvres	5.5	42.5	violet	O	170
Or brûlant	3.2	32.1	orange	N	135
Arbre solitaire	3.6	4.2	rouge	O	17
Lévrier	4	14	or	O	50
Été indien	2.8	7.2	marron	N	30
Grenouille dingue	2.6	21.5	ébène	O	80
Soda et Ça	3.8	4.7	rouge	N	19

Voilà ce à quoi votre table doit ressembler après que vous ayez fait les changements. La vôtre peut être dans un ordre différent mais rappelez-vous, l'ordre n'a pas vraiment de signification.

Ce n'est pas une question piège, mais elle demande réflexion. Si vous changez les boissons à 2,50€ en 3,50€, puis celles de 3,50€ en 4,50€, vous aurez augmenté le prix de la Lune bleue de deux euros. Au lieu de cela, vous devez changer les valeurs les plus grandes d'abord (3,50€ en 4,50€) et ensuite la Lune bleue à 2,50€ en 3,50€.

Encore un de vos exercices-pièges ?

Changez le prix des boissons qui valent 2,50 € en 3,50 € et celui des boissons qui valent actuellement 3,50 € en 4,50 €.

```
INSERT INTO info_boissons VALUES ('Oh mon Dieu', 4.5, 8.6, 'orange', 'O', 35);
DELETE FROM info_boissons WHERE prix = 3.5;
INSERT INTO info_boissons VALUES ('Lune bleue', 3.5, 3.2, 'bleu', 'O', 12),
('Tourbillon de citron vert', 3.5, 5.4, 'vert', 'O', 24);
DELETE FROM info_boissons WHERE prix = 2.5;
```

Des points de bonus si vous avez entré vos deux instructions INSERT en une seule !

Soyez prudent avec votre DELETE

À chaque fois que vous supprimez des enregistrements, vous courez le risque de supprimer accidentellement des enregistrements que vous ne vouliez pas enlever. Par exemple, si nous devons ajouter un nouvel enregistrement pour Bobo :

Utilisez DELETE prudemment. Vérifiez que vous incluez une clause WHERE précise pour viser les lignes exactes que vous voulez supprimer.

Voici l'information que nous devons ajouter et l'INSERT pour le faire.

Bobo vu Chez Valérie

```
INSERT INTO info_clowns
VALUES
('Bobo', 'Chez Valérie', 'H, cigare, cheveux
noirs, chapeau minuscule', 'violon');
```

nom	vu_en_dernier	description	activites
Elsie	Maison de retraite La colline des cerises	F, cheveux rouges, robe verte, pieds immenses	ballons, petite voiture
Cornichon	Fête de Max Leblanc	H, cheveux oranges, costume bleu, pieds immenses	mime
Mme Câlin	La boutique du ballon	F, chemise jaune, pantalon rouge trop grand	klaxon, parapluie
Bobo	Cirque BG	H, cigare, cheveux noirs, chapeau minuscule	violon
Clarabelle	Maison de retraite Belmont	F, cheveux roses, fleur énorme, robe bleue	cris, danse
Scooter	Hôpital Les Chênes	H, cheveux bleus, costume rouge, gros nez	ballons
Zippo	Galerie Pierre du moulin	F, costume orange, pantalon trop grand	danse, chant
Poulette	Les autos de Gilou	F, tout en rose et strass	balançoire, petite voiture
Bozo		H, travesti, robe à pois	chant, danse
Sniff	Chez Valérie	H, costume vert et violet, nez pointu	
Zippo	Galerie Pierre du moulin	F, costume orange, pantalon trop grand	chant
Mme Câlin	La boutique du ballon	F, chemise jaune, pantalon bleu trop grand	klaxon, parapluie
Bozo	Parc Gicard Destin	H, travesti, robe à pois	chant, danse
Sniff	Chez Valérie	H, costume vert et violet, nez pointu	monte dans une petite voiture
Bobo	Fête d'Éric Gris	H, cigare, cheveux noirs, chapeau minuscule	violon
Bobo	Chez Valérie	H, cigare, cheveux noirs, chapeau minuscule	violon

INSERT

Maintenant à vous d'être le DELETE

Vous ÊTES le DELETE

Vous trouverez ci-dessous une série de clauses WHERE pour une instruction DELETE conçue pour nettoyer la table info_clowns de la page ci-contre. Identifiez celles qui nous aident et celles qui créent de nouveaux problèmes.

Cela nous aide-t-il ? Si non, dites pourquoi.

```
DELETE FROM info_clowns

  WHERE vu_en_dernier = 'Hôpital Les Chênes';

  WHERE activites = 'violon';

  WHERE vu_en_dernier = 'Parc Gicard Destin'
  AND nom = 'Bobo';

  WHERE vu_en_dernier = 'Hôpital Les Chênes'
  AND vu_en_dernier = 'Parc Gicard Destin';

  WHERE vu_en_dernier = 'Hôpital Les Chênes'
  OR vu_en_dernier = 'Parc Gicard Destin';

  WHERE nom = 'Bobo'
  OR vu_en_dernier = 'Hôpital Les Chênes';
```

Écrivez maintenant une instruction DELETE unique qui puisse supprimer tous les enregistrements en trop pour Bobo sans en toucher aucun autre.

Vous ÊTES le DELETE - Solution

Vous trouverez ci-dessous une série de clauses WHERE pour une instruction DELETE conçue pour nettoyer la table info_clowns de la page ci-contre. Identifiez celles qui nous aident et celles qui créent de nouveaux problèmes.

Cela nous aide-t-il ? Si non, dites pourquoi.

```
DELETE FROM info_clowns
```

Scooter a une ligne qui correspond à ceci.

```
WHERE vu_en_dernier = 'Hôpital Les Chênes';
```

Supprime l'enregistrement de Scooter.

Nous ne voulons pas supprimer le nouvel enregistrement

```
WHERE activites = 'violon';
```

Supprime tous les enregistrements de Bobo, y compris le nouveau.

```
WHERE vu_en_dernier = 'Parc Gicard Destin'
AND nom = 'Bobo';
```

Ne supprime rien.

Le AND signifie que les deux doivent être vrais.

```
WHERE vu_en_dernier = 'Hôpital Les Chênes'
AND vu_en_dernier = 'Parc Gicard Destin';
```

Ne supprime rien.

```
WHERE vu_en_dernier = 'Hôpital Les Chênes'
OR vu_en_dernier = 'Parc Gicard Destin';
```

Supprime les enregistrements de Bozo et de Scooter.

```
WHERE nom = 'Bobo'
OR vu_en_dernier = 'Hôpital Les Chênes';
```

Supprime tous les enregistrements de Bobo y compris le nouveau et supprime celui de Scooter.

Écrivez maintenant une instruction DELETE unique qui puisse supprimer tous les enregistrements en trop pour Bobo sans en toucher aucun autre.

```
DELETE FROM info_clowns
WHERE nom = 'Bobo'
AND vu_en_dernier <> 'Chez Valérie';
```

Exact ! À moins d'être absolument certain que votre clause WHERE va supprimer les lignes que vous voulez, vous devriez utiliser un SELECT d'abord pour vérifier.

Comme ils utilisent tous deux la même clause `WHERE`, les lignes renvoyées par le `SELECT` reflèteront celles que vous supprimerez par un `DELETE` avec cette clause `WHERE`.

C'est un moyen sans danger pour vérifier que vous ne supprimez rien par accident et cela vous permettra d'être sûr de supprimer tous les enregistrements que vous voulez.

Le problème avec un DELETE imprécis

DELETE est dangereux. Si nous ne faisons pas attention, les mauvaises données seront visées. Nous pouvons éviter de prendre pour cible les mauvaises données si nous ajoutons une autre étape à nos deux étapes INSERT-DELETE.

Voici un plan en TROIS ÉTAPES que nous pouvons suivre :

Ne modifiez que les enregistrements que vous voulez changer en utilisant l'instruction SELECT.

1 D'abord, insérez le nouvel enregistrement avec INSERT.

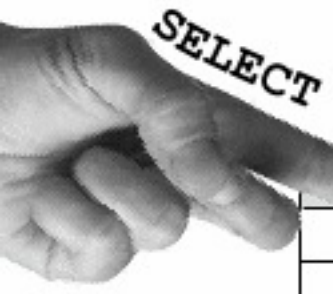
```
INSERT INTO info_clowns
VALUES
('Zippo', 'Galerie Pierre du moulin', 'F, costume
orange, pantalon trop grand', 'danse, chant');
```

Insérez l'enregistrement en utilisant les données d'origine et en modifiant uniquement la colonne que vous avez besoin de changer.

nom	vu_en_dernier	description	activites
Zippo	Galerie Pierre du moulin	F, costume orange, pantalon trop grand	danse
Zippo	Galerie Pierre du moulin	F, costume orange, pantalon trop grand	danse, chant

2 Puis, SELECTionnez l'enregistrement qui doit être enlevé pour vérifier que vous allez supprimer le bon enregistrement et aucun des autres.

```
SELECT FROM info_clowns
WHERE
activites = 'danse';
```

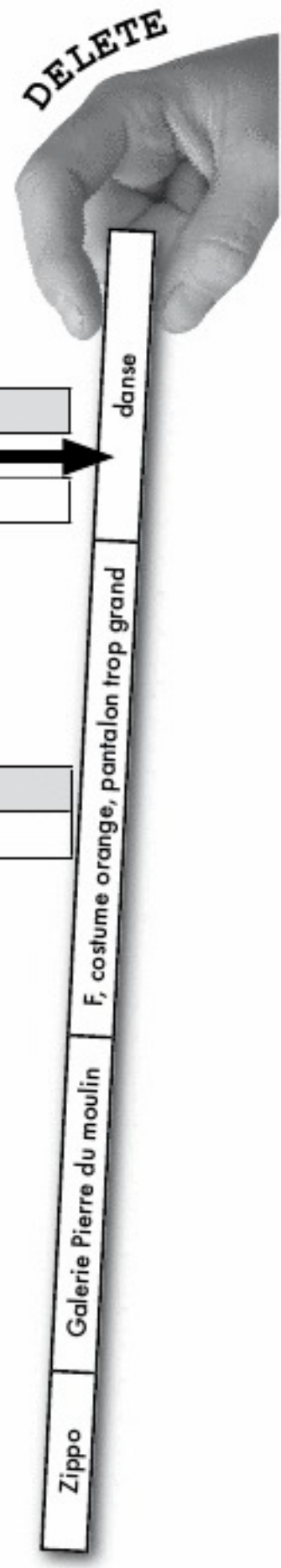


nom	vu_en_dernier	description	activites
Zippo	Galerie Pierre du moulin	F, costume orange, pantalon trop grand	danse
Zippo	Galerie Pierre du moulin	F, costume orange, pantalon trop grand	danse, chant

3 Enfin, supprimez les anciens enregistrements en utilisant `DELETE` et la même clause `WHERE` que vous avez utilisée avec votre `SELECT` dans la deuxième étape.

```
DELETE FROM info_clowns
WHERE
activites = 'danse';
```

Utilisez la clause WHERE que vous avez utilisée pour SELECTionner l'enregistrement dans la nouvelle étape 2 pour trouver et supprimer l'ancien enregistrement avec DELETE.



nom	vu_en_dernier	description	activites
Zippo	Galerie Pierre du moulin	F, costume orange, pantalon trop grand	danse, chant

Maintenant il ne nous reste plus que le nouvel enregistrement.

nom	vu_en_dernier	description	activites
Zippo	Galerie Pierre du moulin	F, costume orange, pantalon trop grand	danse, chant

Est-ce que ce ne serait pas merveilleux si je pouvais changer un enregistrement en une seule étape sans m'inquiéter que mon nouvel enregistrement puisse être détruit en même temps que l'ancien. Mais ce n'est sans doute qu'un rêve...

Modifiez vos données avec UPDATE

Vous devriez maintenant être à l'aise avec l'utilisation d'`INSERT` et de `DELETE` pour garder vos tables à jour et nous avons vu certaines façons de les utiliser ensemble pour modifier indirectement une ligne donnée.

Mais plutôt que d'insérer une nouvelle ligne et de supprimer l'ancienne, vous pouvez redéfinir, ou réutiliser, une ligne qui se trouve déjà dans votre table en ne changeant que les valeurs des colonnes que vous voulez changer.

L'instruction SQL s'appelle `UPDATE` et elle fait exactement ce par quoi elle se traduit. Elle met à jour une colonne ou des colonnes avec une nouvelle valeur et tout comme pour `SELECT` et `DELETE`, vous pouvez lui ajouter une clause `WHERE` pour indiquer quelle ligne vous voulez mettre à jour avec votre `UPDATE`.

Voici `UPDATE` à l'œuvre :

```
UPDATE evaluation_crepes
SET
type = 'sucre'
WHERE type = 'sucre glace';
```

C'est là que nous indiquons la nouvelle valeur.

Voici une clause WHERE standard, tout comme celles que vous avez vues avec SELECT et DELETE.

Le mot-clé `SET` dit au SGBDR qu'il doit changer la colonne située avant le signe égal pour qu'elle contienne la valeur située après le signe égal. Dans le cas ci-dessus, nous changeons `'sucre glace'` en `'sucre'` dans notre table. Le `WHERE` dit de ne changer que les lignes où le type est `'sucre glace'`.

evaluation_crepes

lieu	heure	date	type	note	commentaires
Le roi de la croustille	8:50	9/27	sucre glace	10	presque parfaite
La crêperie du port	8:59	8/25	NULL	6	grasse
Café Starbuzz	19:35	5/24	tarte au citron	5	tiède mais savoureuse
La crêperie du port	19:03	4/26	confiture	7	manque de confiture

evaluation_crepes

lieu	heure	date	type	note	commentaires
Le roi de la croustille	8:50	9/27	sucre	10	presque parfaite
La crêperie du port	8:59	8/25	NULL	6	grasse
Café Starbuzz	19:35	5/24	tarte au citron	5	tiède mais savoureuse
La crêperie du port	19:03	4/26	confiture	7	manque de confiture

Les règles de l'UPDATE

- Vous pouvez utiliser `UPDATE` pour changer la valeur d'une seule colonne ou d'une série de colonnes. Ajoutez plus de paires `colonne = valeur` à la clause SET et placez une virgule après chacune d'elle :

```
UPDATE votre_table
SET premiere_colonne = 'nouvellevaleur',
seconde_colonne = 'autre_valeur';
```

- Vous pouvez utiliser `UPDATE` pour mettre à jour une seule ligne ou plusieurs lignes, selon la clause `WHERE`.

il n'y a pas de Questions stupides

Q: Que se passe-t-il si j'enlève la clause WHERE ?

R: Alors toutes les colonnes de la clause SET de votre table seront mises à jour avec la nouvelle valeur.

Q: Il y a deux signes égal dans la requête SQL de la page de gauche qui semblent faire des choses différentes. Est-ce correct ?

R: Tout à fait. Le signe égal dans la clause SET indique « rend cette colonne égale à cette valeur » alors que celle de la clause WHERE teste si la valeur de la colonne est égale à la valeur qui suit le signe.

Q: Puis-je utiliser cette instruction pour faire la même chose que page de gauche ?

```
UPDATE evaluation_crepes SET type
= 'sucre' WHERE lieu = 'Le roi de la
croustille';
```

R: Oui, vous le pouvez. Cela mettrait à jour la même ligne de la même façon et cela ne pose aucun problème avec notre table de quatre lignes. Mais si vous aviez utilisé cela pour une table contenant des centaines ou des milliers d'enregistrements, vous auriez changé le type pour chacune des lignes du roi de la croustille.

Q: Oups ! Comment être sûr de ne mettre à jour que ce que je veux ?

R: De la même façon qu'avec DELETE : à moins d'être certain que vous ciblez les lignes correctes avec votre clause WHERE, faites d'abord un SELECT !

Q: Peut-on avoir plus d'une clause SET ?

R: Non mais vous ne devriez pas en avoir besoin. Vous pouvez mettre toutes vos colonnes et leurs nouvelles valeurs dans la même clause SET, comme expliqué ci-dessus.

UPDATE est le nouvel INSERT-DELETE

Quand vous utilisez **UPDATE**, vous ***ne supprimez rien***. Au lieu de cela, vous **recyclez l'ancien enregistrement pour en faire un nouveau**.

Commencez par UPDATE...

...puis le nom de la table contenant l'enregistrement que vous voulez utiliser.

```
UPDATE nom_table
SET nom_colonne = nouvellevaleur
WHERE nom_colonne = cettevaleur;
```

SET indique la modification que vous allez faire à l'enregistrement.

Notre bonne vieille clause WHERE est là pour nous aider à cibler précisément l'enregistrement à changer.

Les instructions UPDATE peuvent remplacer les combinaisons INSERT/DELETE.

Voyons tout cela en action avec une commande qui va fonctionner pour la table `info_clowns`.

Change la valeur de la colonne vu_en_dernier en Chez Valérie.

UPDATE met à jour un enregistrement dans la table info_clowns.

```
UPDATE info_clowns
SET vu_en_dernier = 'Chez Valérie'
WHERE nom = 'Bobo'
AND vu_en_dernier = 'Fête d\'Éric Gris';
```

Voici la clause WHERE pour indiquer précisément l'enregistrement à changer : dans ce cas, l'enregistrement de Bobo avec la valeur Fête d'Éric Gris pour vu_en_dernier.

UPDATE à l'œuvre

En utilisant l'instruction `UPDATE`, la colonne `vu_en_dernier` de l'enregistrement de Bobo est modifiée de Fête d'Éric Gris en Chez Valérie.

Bobo vu Chez Valérie

Voici l'information que nous avons besoin d'ajouter et l'UPDATE que nous allons utiliser pour le faire.

```
UPDATE info_clowns
SET vu_en_dernier = 'Chez Valérie'
WHERE nom = 'Bobo'
AND vu_en_dernier = 'Fête d\'Éric Gris';
```

nom	vu_en_dernier	description	activites
Elsie	Maison de retraite La colline des cerises	F, cheveux rouges, robe verte, pieds immenses	ballons, petite voiture
Cornichon	Fête de Max Leblanc	H, cheveux oranges, costume bleu, pieds immenses	mime
Mme Câlin	La boutique du ballon	F, chemise jaune, pantalon rouge trop grand	klaxon, parapluie
Bobo	Cirque BG	H, cigare, cheveux noirs, chapeau minuscule	violon
Clarabelle	Maison de retraite Belmont	F, cheveux roses, fleur énorme, robe bleue	cris, danse
Scooter	Hôpital Les Chênes	H, cheveux bleus, costume rouge, gros nez	ballons
	Galerie Pierre du moulin	F, costume orange, pantalon trop grand	danse, chant
	Les autos de Gilou	F, tout en rose et strass	balançoire, petite voiture
		H, travesti, robe à pois	chant, danse
	Chez Valérie	H, costume vert et violet, nez pointu	
	Galerie Pierre du moulin	F, orange, pantalon grand	chant
	La boutique du ballon	trop grand	klaxon, parapluie
		be à pois	chant, danse
		et, nez pointu	monte dans une petite voiture
Bobo	Chez Valérie	eau minuscule	violon

UPDATE

En utilisant UPDATE, vous éditez cet endroit, il n'y a donc aucun risque de supprimer des données sans le vouloir (mais vous remplacez quand même des données existantes).

À vos crayons !

Mise à jour des déplacements des clowns

Cette fois, faisons bien les choses. Écrivez une instruction UPDATE pour chaque rapport. Nous en avons fait une pour vous aider à commencer. Puis complétez la table info_clowns pour qu'elle ressemble à celle que nous obtiendrons après avoir exécuté toutes ces instructions UPDATE.

Zippo vu en train de chanter

```
UPDATE info_clowns
SET activites = 'chant'
WHERE nom = 'Zippo';
```

Mme Câlin porte maintenant un pantalon bleu trop grand

Bozo vu au Parc Gicard Destin

Sniff monte dans une petite voiture

Bobo vu récemment à la fête d'Éric Gris

nom	vu_en_dernier	description	activites
Elsie	Maison de retraite La colline des cerises	F, cheveux rouges, robe verte, pieds immenses	ballons, petite voiture
Cornichon	Fête de Max Leblanc	H, cheveux oranges, costume bleu, pieds immenses	mime
Mme Câlin	La boutique du ballon	F, chemise jaune, pantalon rouge trop grand	klaxon, parapluie
Bobo	Cirque BG	H, cigare, cheveux noirs, chapeau minuscule	violon
Clarabelle	Maison de retraite Belmont	F, cheveux roses, fleur énorme, robe bleue	cris, danse
Scooter	Hôpital Les Chênes	H, cheveux bleus, costume rouge, gros nez	ballons
Zippo	Galerie Pierre du moulin	F, costume orange, pantalon trop grand	danse
Poulette	Les autos de Gilou	F, tout en rose et strass	balançoire, petite voiture
Bozo		H, travesti, robe à pois	chant, danse
Sniff	Chez Valérie	H, costume vert et violet, nez pointu	

nom	vu_en_dernier	description	activites
Elsie	Maison de retraite La colline des cerises	F, cheveux rouges, robe verte, pieds immenses	ballons, petite voiture
Cornichon	Fête de Max Leblanc	H, cheveux oranges, costume bleu, pieds immenses	mime
Mme Câlin			
Bobo			
Clarabelle	Maison de retraite Belmont	F, cheveux roses, fleur énorme, robe bleue	cris, danse
Scooter	Hôpital Les Chênes	H, cheveux bleus, costume rouge, gros nez	ballons
Zippo			
Poulette	Les autos de Gilou	F, tout en rose et strass	balançoire, petite voiture
Bozo			
Sniff			

À vos crayons !

Mise à jour des déplacements des clowns

Votre mission était d'écrire une instruction UPDATE pour chaque rapport puis de compléter la table info_clowns pour qu'elle ressemble à celle que nous obtiendrons après avoir exécuté toutes ces instructions UPDATE.

Zippo vu en train de chanter

```
UPDATE info_clowns
SET activites = 'chant'
WHERE nom = 'Zippo';
```

Mme Câlin porte maintenant un pantalon bleu trop grand

Nous ne voulons pas supprimer les autres informations qui sont déjà dans la colonne description. Vérifiez que vous les avez incluses ici.

```
UPDATE info_clowns
SET description = 'F, chemise jaune, pantalon bleu trop grand'
WHERE nom = 'Mme Câlin';
```

Bozo vu au Parc Gicard Destin

```
UPDATE info_clowns
SET vu_en_dernier = 'Parc Gicard Destin'
WHERE nom = 'Bozo';
```

Sniff monte dans une petite voiture

```
UPDATE info_clowns
SET activites = 'monte dans une petite voiture'
WHERE nom = 'Sniff';
```

Bobo vu récemment à la fête d'Éric Gris

N'oubliez pas d'échapper aux apostrophes dans vos valeurs VARCHAR.

```
UPDATE info_clowns
SET vu_en_dernier = 'Fête d\'Éric Gris'
WHERE nom = 'Bobo';
```

nom	vu_en_dernier	description	activites
Elsie	Maison de retraite La colline des cerises	F, cheveux rouges, robe verte, pieds immenses	ballons, petite voiture
Cornichon	Fête de Max Leblanc	H, cheveux oranges, costume bleu, pieds immenses	mime
Mme Câlin	La boutique du ballon	F, chemise jaune, pantalon rouge trop grand	klaxon, parapluie
Bobo	Cirque BG	H, cigare, cheveux noirs, chapeau minuscule	violon
Clarabelle	Maison de retraite Belmont	F, cheveux roses, fleur énorme, robe bleue	cris, danse
Scooter	Hôpital Les Chênes	H, cheveux bleus, costume rouge, gros nez	ballons
Zippo	Galerie Pierre du moulin	F, costume orange, pantalon trop grand	danse
Poulette	Les autos de Gilou	F, tout en rose et strass	balançoire, petite voiture
Bozo		H, travesti, robe à pois	chant, danse
Sniff	Chez Valérie	H, costume vert et violet, nez pointu	

Les enregistrements en gris n'ont pas changé parce que nous ne les avons pas mis à jour.

nom	vu_en_dernier	description	activites
Elsie	Maison de retraite La colline des cerises	F, cheveux rouges, robe verte, pieds immenses	ballons, petite voiture
Cornichon	Fête de Max Leblanc	H, cheveux oranges, costume bleu, pieds immenses	mime
Mme Câlin	La boutique du ballon	F, chemise jaune, pantalon bleu trop grand	klaxon, parapluie
Bobo	Fête d'Éric Gris	H, cigare, cheveux noirs, chapeau minuscule	violon
Clarabelle	Maison de retraite Belmont	F, cheveux roses, fleur énorme, robe bleue	cris, danse
Scooter	Hôpital Les Chênes	H, cheveux bleus, costume rouge, gros nez	ballons
Zippo	Galerie Pierre du moulin	F, costume orange, pantalon trop grand	chant
Poulette	Les autos de Gilou	F, tout en rose et strass	balançoire, petite voiture
Bozo	Parc Gicard Destin	H, travesti, robe à pois	chant, danse
Sniff	Chez Valérie	H, costume vert et violet, nez pointu	monte dans une petite voiture

Seules les parties de chaque enregistrement que nous avons définies avec SET dans l'UPDATE ont changé. Nous avons finalement complété ces blancs qui remontent à la page 121.

UPDATE met à jour vos prix

Vous souvenez-vous du moment où nous avons essayé de changer des prix dans la table `info_boissons` ? Nous voulions changer les prix des boissons à 2,50 € en 3,50 € et celui des boissons à 3,50 € en 4,50 €.

info_boissons

nom_boisson	prix	carbs	couleur	glace	calories
Épine noire	3	8.4	jaune	O	33
Lune bleue	2.5	3.2	bleu	O	12
Oh mon Dieu	3.5	8.6	orange	O	35
Tourbillon de citron vert	2.5	5.4	vert	O	24
Baiser sur les lèvres	5.5	42.5	violet	O	171
Or brûlant	3.2	32.1	orange	N	135
Arbre solitaire	3.6	4.2	rouge	O	17
Lévrier	4	14	jaune	O	50
Été indien	2.8	7.2	marron	N	30
Grenouille dingue	2.6	21.5	ébène	O	80
Soda et Ça	3.8	4.7	rouge	N	19

Voyons comment nous pouvons aborder ce problème en utilisant une instruction `UPDATE` pour nous occuper de chaque enregistrement individuellement en écrivant une série d'instructions `UPDATE` comme celle-ci :

```
UPDATE info_boissons
SET prix = 3.5
WHERE nom_boisson = 'Lune bleue';
```

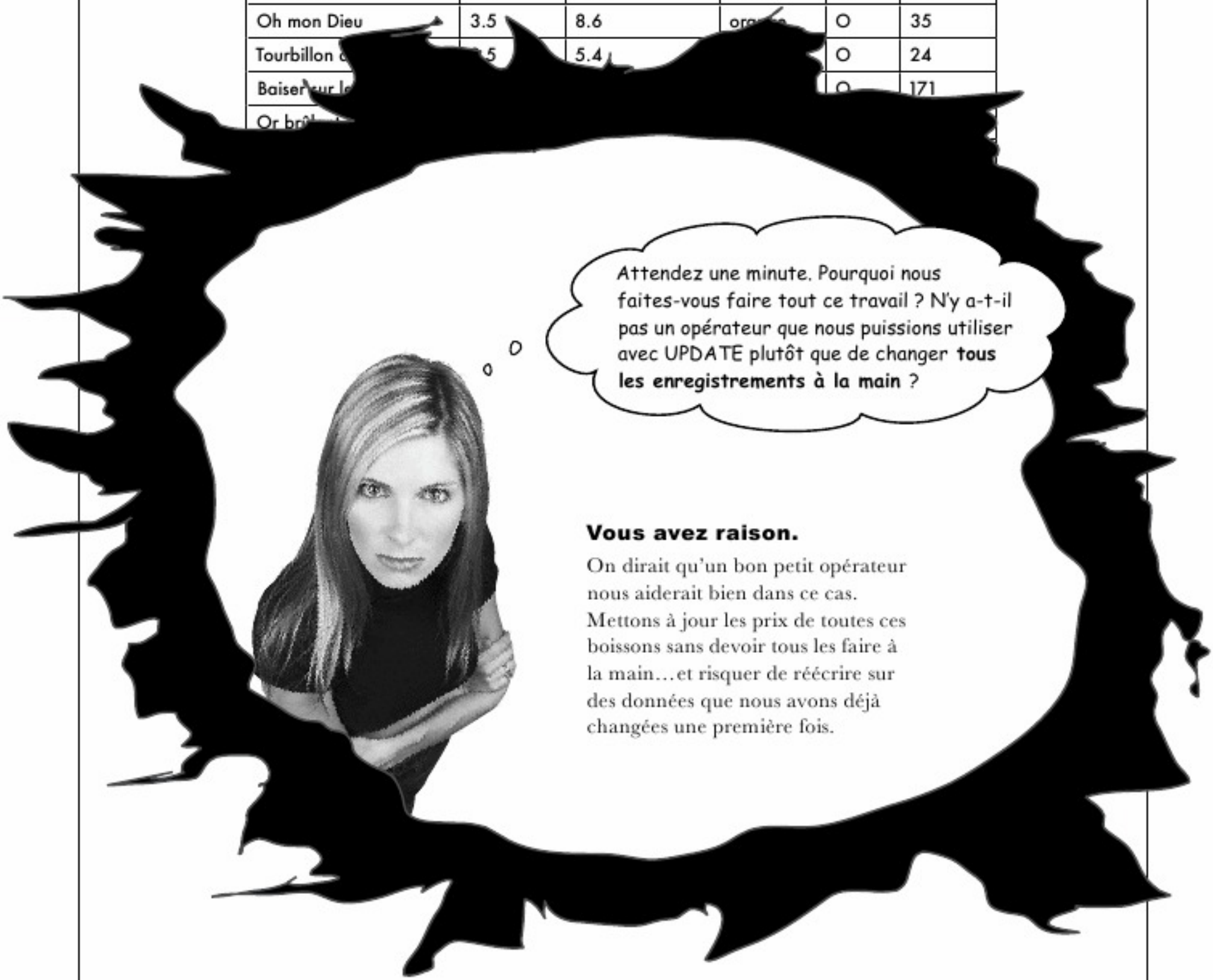
Le prix avec 1 € en plus.

Nous utilisons un WHERE pour choisir une seule colonne, ainsi nous savons quel enregistrement est mis à jour.

À vos crayons !

Écrivez des instructions `UPDATE` pour chaque enregistrement de la table `info_boissons` pour ajouter un euro à tous les prix.

nom_boisson	prix	carbs	couleur	glace	calories
Épine noire	3	8.4	jaune	O	33
Lune bleue	2.5	3.2	bleu	O	12
Oh mon Dieu	3.5	8.6	[illegible]	O	35
Tourbillon [illegible]	[illegible]	5.4	[illegible]	O	24
Baiser sur le [illegible]	[illegible]	[illegible]	[illegible]	O	171
Or brû[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]



Vous avez raison.

On dirait qu'un bon petit opérateur nous aiderait bien dans ce cas. Mettons à jour les prix de toutes ces boissons sans devoir tous les faire à la main…et risquer de réécrire sur des données que nous avons déjà changées une première fois.

Nous n'avons besoin que d'un UPDATE

Notre colonne prix est un nombre. En SQL, nous pouvons réaliser des opérations de ***maths élémentaires*** sur les **colonnes de nombres**. Dans le cas de notre colonne de prix, nous pouvons simplement y ajouter 1 pour toutes les lignes de notre table que nous devons changer. Voici comment :

```
UPDATE info_boissons
SET prix = prix + 1
WHERE
nom_boisson='Lune bleue'
OR
nom_boisson='Oh mon Dieu'
OR
nom_boisson= 'Tourbillon de
citron vert';
```

Ajoute 1 à chacun des trois prix (les boissons à 2,50 € et 3,50 €) que nous avons besoin de changer.

Questions stupides

il n'y a pas de

Q: Puis-je utiliser la soustraction avec une valeur numérique ? Que puis-je utiliser d'autre ?

R: Multiplication, division, soustraction, vous pouvez toutes les utiliser et vous pouvez réaliser ces opérations avec d'autres valeurs numériques, pas qu'avec 1.

Q: Pouvez-vous me donner un exemple d'une situation où je pourrais utiliser une multiplication ?

R: Bien sûr. Supposez que vous ayez une liste de produits dans votre table, chacun ayant un prix. Vous pouvez utiliser une instruction `UPDATE` et multiplier tous les prix avec un nombre fixe pour calculer le prix de ces produits en incluant la TVA.

Q: Alors, y a-t-il d'autres opérations que l'on peut réaliser avec les données en plus des maths simples ?

R: Il y en a pas mal. Nous parlerons un peu plus tard de ce que vous pouvez faire avec vos variables texte et de ce que vous pouvez faire d'autre avec les numériques.

Q: Comme quoi ? Donnez-nous un indice.

R: D'accord. Par exemple, vous pouvez utiliser la fonction `UPPER()` pour passer une colonne de texte toute entière en majuscules. Et logiquement, `LOWER()` passe tout en minuscules.

Les instructions UPDATE peuvent s'utiliser sur plusieurs enregistrements de votre table. Utilisez-les avec les opérateurs de maths élémentaires pour manipuler vos valeurs numériques.

Les données changent, savoir comment les mettre à jour est donc crucial.

Toutefois, mieux vous concevez votre table, moins vous aurez à faire de mises à jour globales. Une bonne conception de table vous soulage et vous permet de vous concentrer sur les données de la table.

Intéressé ? Nous allons maintenant regarder de plus près une manière inattendue de concevoir des tables...

Boîte à outils SQL

Le chapitre 3 ne sera bientôt plus qu'un souvenir. Mais voici un rapide résumé des nouvelles instructions SQL que vous avez apprises. Pour une liste exhaustive des outils du livre, voyez l'annexe C.

DELETE

C'est l'outil pour supprimer des lignes de données de votre table. Utilisez-le avec une clause WHERE pour cibler précisément les lignes que vous voulez enlever.

UPDATE

Cette instruction met à jour une ou plusieurs colonnes avec une nouvelle valeur. Elle utilise également une clause WHERE.

SET

Ce mot-clé fait partie d'une instruction UPDATE et s'utilise pour modifier la valeur d'une colonne existante.

À quoi bon être normal ?

Vous avez créé des tables sans leur accorder trop de réflexion. Et tout va bien, elles fonctionnent. Vous pouvez utiliser `SELECT`, `INSERT`, `DELETE` et `UPDATE`. Mais **plus vous obtenez de données**, plus vous commencez à voir des **choses que vous auriez aimé avoir fait** pour simplifier vos clauses `WHERE`. Vous avez besoin de rendre vos tables plus *normales*.

Deux tables poissonneuses

Jacques et Marc ont tous les deux créé des tables pour conserver des informations sur les plus belles prises de poissons. La table de Marc comporte des colonnes pour le nom commun et l'espèce du poisson, son poids et le lieu où il a été pris. Elle ne comporte pas le nom des personnes ayant attrapé le poisson.

info_poissons

nom_commun	espece	lieu	poids
bar	D. Labrax	Saint Brieuc, 22	10,100 kg
sandre	S. Lucioperca	Lac de Chemillé sur Indrois, 37	11,360 kg
truite fario	S. T. Fario	Lac du Monteynard, 38	18,600 kg
perche jaune	P. Flavescens	Laneuville, 54	1,900 kg
dorade royale	S. Aurata	Bassin d'Arcachon, 33	2,100 kg
barbeau	B. Barbus	Chambilly, 71	22,800 kg
anguille	A. Anguilla	Thionville, 57	2,350 kg
poisson rouge	C. Auratus	Pertuis, 84	0,450 kg
raie pastenague	D. Pastinaca	Bassin d'Arcachon, 33	2,900 kg
saumon royal	O. Tshawytscha	Dordogne, 24	44,100 kg

Cette table n'a que quatre colonnes. Comparez-la à la table prises_records en face.

La table de Jacques comporte le nom commun et le poids du poisson, mais également le prénom et le nom de famille de la personne l'ayant attrapé. De plus le lieu est divisé en deux colonnes, l'une pour le nom du lieu où le poisson a été pris et l'autre pour le département.

Cette table concerne aussi les prises records mais elle a presque deux fois plus de colonnes.

prises_records

prenom	nom	nom_commun	lieu	dpmt	poids	date
Georges	Baudry	bar	Saint Brieuc	22	10,100 kg	02/06/1932
Soufiane	Sagbo	sandre	Lac de Chemillé sur Indrois	37	11,360 kg	02/08/1960
Yves	Cholet	truite fario	Lac du Monteynard	38	18,600 kg	01/12/1925
Raphaël	Gérard	perche jaune	Laneuville	54	1,900 kg	01/05/1865
Gilbert	Pillot	dorade royale	Bassin d'Arcachon	33	8,100 kg	09/04/1950
Jean-Marie	Teboul	barbeau	Chambilly	71	5,800 kg	30/07/1954
Fred	Perron	anguille	Thionville	57	2,350 kg	31/07/1957
Michel	Berg	poisson rouge	Pertuis	84	0,450 kg	09/06/1990
Alain	Abena	raie pastenague	Bassin d'Arcachon	33	2,900 kg	17/04/1996
Hervé	Leroy	saumon royal	Dordogne	24	44,100 kg	17/05/1985

À vos crayons !

J'écris pour le magazine *Tourne le moulinet*. Je dois connaître les noms des pêcheurs, les dates et les endroits des plus belles prises.

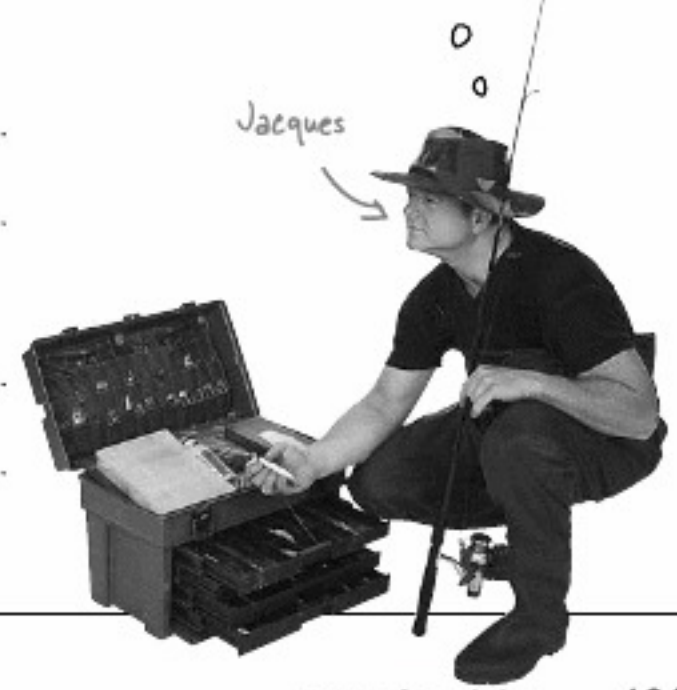
Écrivez une requête pour **chacune des tables** pour trouver tous les records du 54.

.....

.....

.....

.....



À vos crayons ! Solution

Écrivez une requête pour **chacune des tables** pour trouver tous les records du 54.

Je n'ai presque jamais besoin de chercher par département. J'ai inséré les données avec les départements dans la même colonne que la ville.

Nous devons utiliser LIKE pour obtenir nos résultats à partir de la ville et du département combinés.

```
SELECT * FROM info_poissons
WHERE lieu LIKE '%54';
```

nom_commun	espece	lieu	poids
perche jaune	P. Flavescens	Laneuville, 54	1,900 kg

Je dois souvent faire des recherches par département, alors je les ai mis dans une colonne dpmt séparée quand j'ai créé ma table.

Cette requête peut chercher directement dans la colonne dpmt.

```
SELECT * FROM prises_records
WHERE dpmt = '54';
```

prenom	nom	nom_commun	lieu	dpmt	poids	date
Raphaël	Gérard	perche jaune	Laneuville	54	1,900 kg	01/05/1865

il n'y a pas de

Questions stupides

Q: La table de Jacques est-elle meilleure que celle de Marc ?

R: Non. Ce sont des tables différentes répondant à des objectifs différents. Marc aura rarement besoin de chercher directement un département parce qu'il ne s'intéresse vraiment qu'au nom vulgaire et à l'espèce des plus gros poissons ainsi qu'à leur poids.

Jacques, par contre, *aura* besoin de chercher des départements quand il consultera ses données. C'est pourquoi sa table comporte une colonne séparée : pour lui permettre de cibler facilement les départements dans ses recherches.

Q: Est-ce que l'on doit éviter LIKE quand on fait des requêtes dans nos tables ? Est-ce qu'il fonctionne mal ?

R: `LIKE` fonctionne très bien mais il peut être difficile à utiliser dans vos requêtes et vous risquez d'obtenir des résultats dont vous ne voulez pas. Si vos colonnes contiennent des informations complexes, `LIKE` n'est pas assez spécifique pour cibler des données précises.

Q: Pourquoi les requêtes courtes sont-elles meilleures que les longues ?

R: Plus la requête est simple, mieux c'est. Au fur et à mesure que votre base de données grandit et que vous ajoutez de nouvelles tables, vos requêtes se compliquent. Si vous commencez avec la requête la plus simple possible maintenant, vous l'apprécierez plus tard.

Q: Est-ce que cela signifie que je dois toujours avoir des toutes petites parties de données dans mes colonnes ?

R: Pas nécéssairement. Comme vous commencez à le voir avec les tables de Marc et de Jacques, cela dépend de la façon dont vous allez ***utiliser*** les données.

Par exemple, imaginez une table de voitures pour un mécanicien et une pour un concessionnaire. Le mécanicien peut avoir besoin d'informations précises sur chaque voiture mais le vendeur voudra seulement la marque de la voiture, le modèle et le numéro VIN.

Q: Supposez que nous ayons une adresse avec un nom de rue. Pourquoi ne pourrions-nous pas avoir une colonne avec l'adresse complète, puis d'autres colonnes avec les données séparées ?

R: Même si pour l'instant dupliquer vos données peut sembler une bonne idée, pensez à la place que cela va prendre sur votre disque dur quand votre base de données sera énorme. De plus, avec chaque duplicat de vos données, vous devez ajouter une clause de plus à votre instruction UPDATE quand vos données changent.

Voyons de plus près comment concevoir vos tables de la façon la plus adaptée à vos besoins.

La façon dont vous allez utiliser vos données influence la façon de définir votre table.

Musclez vos neurones

SQL est le langage utilisé par les bases de données relationnelles. À votre avis, que signifie « relationnel » dans une base de données SQL ?

Une table est centrée sur ses relations

SQL est connu pour être un Système de Gestion de Base de Données Relationnelles ou SGBDR. Vous n'avez pas besoin de le retenir. Nous ne nous intéressons qu'au mot RELATIONNELLES*. Tout ce que cela signifie pour vous, c'est que pour concevoir une super table, vous devez prendre en compte **la façon dont** les **colonnes *sont reliées*** les unes aux autres **pour décrire une chose**.

Le défi est de décrire cette chose en utilisant des colonnes d'une façon qui facilite la récupération des informations. Cela dépend de ce que vous voulez de la table, mais il existe des étapes très générales que vous pouvez suivre quand vous créez une table.

1. **Prenez une chose, la chose particulière que vous voulez que votre table décrive.**

Quelle est la chose principale sur laquelle vous voulez centrer votre table ?

2. **Faites une liste des informations que vous avez besoin de connaître sur cette chose quand vous utilisez la table.**

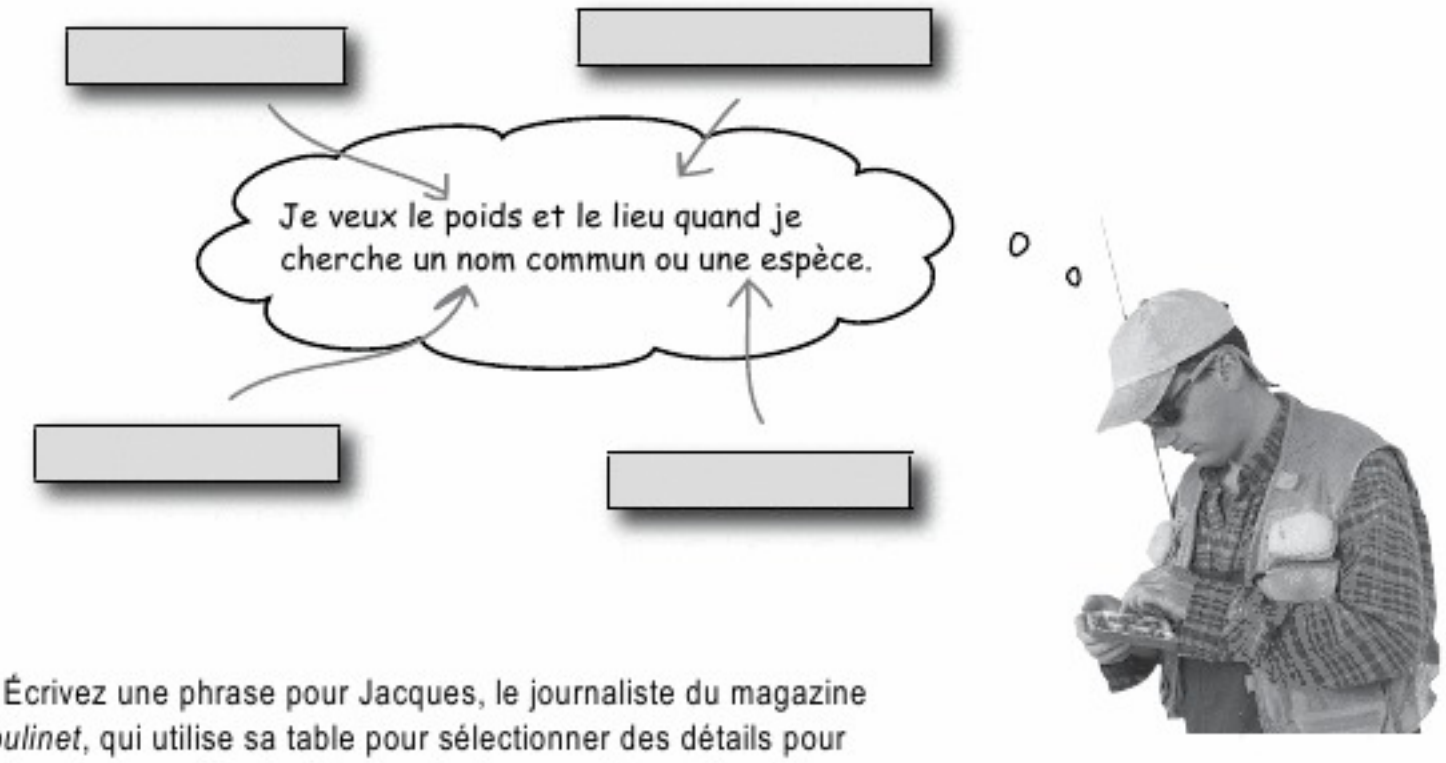
Comment allez-vous utiliser cette table ?

3. **En utilisant la liste, divisez les informations sur votre chose en morceaux que vous pouvez utiliser pour organiser votre table.**

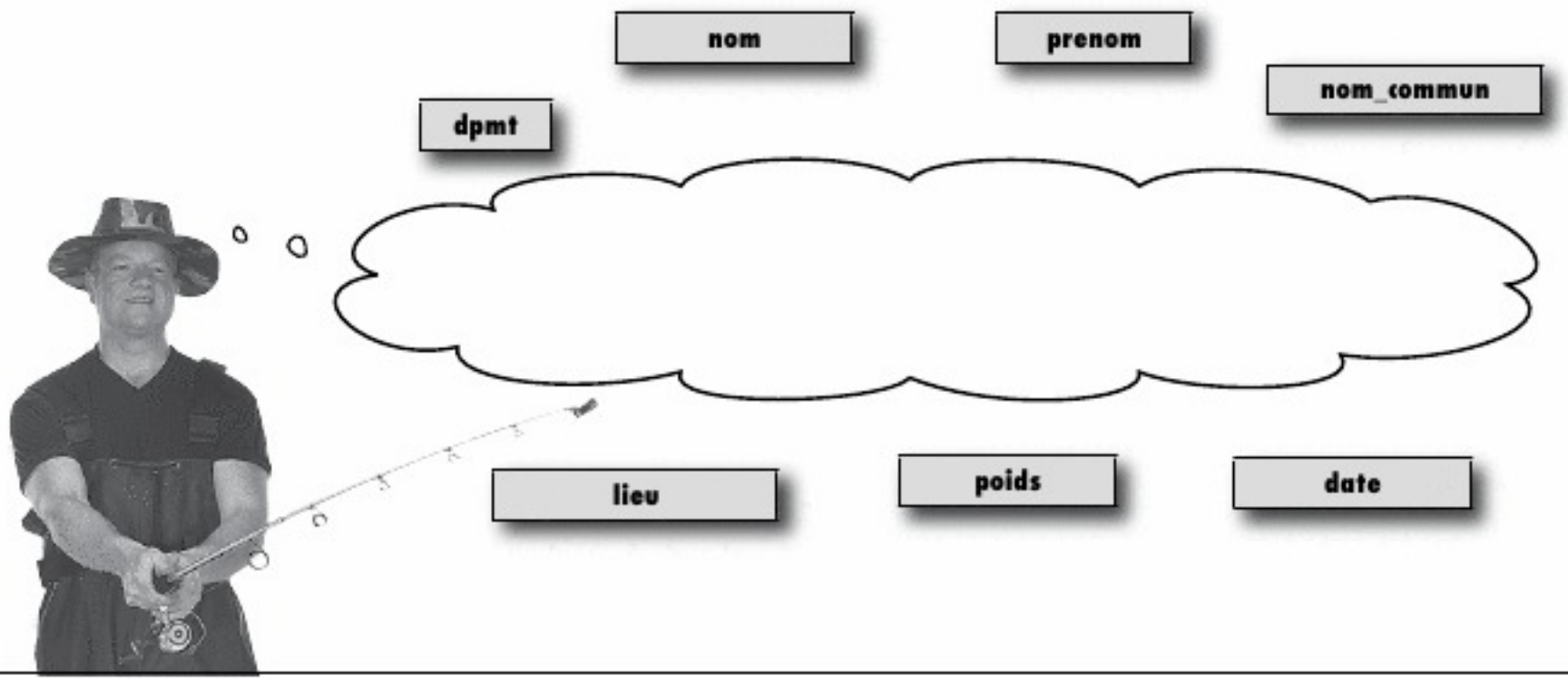
Comment pouvez-vous interroger cette table le plus facilement ?

* Certains pensent que RELATIONNELLES signifie plusieurs *tables* reliées les unes aux autres. Ce n'est pas ça.

Pouvez-vous repérer dans cette phrase les colonnes qu'a utilisé Marc l'ichthyologue pour décrire la façon dont il veut interroger sa table ? Complétez avec les noms de colonnes.

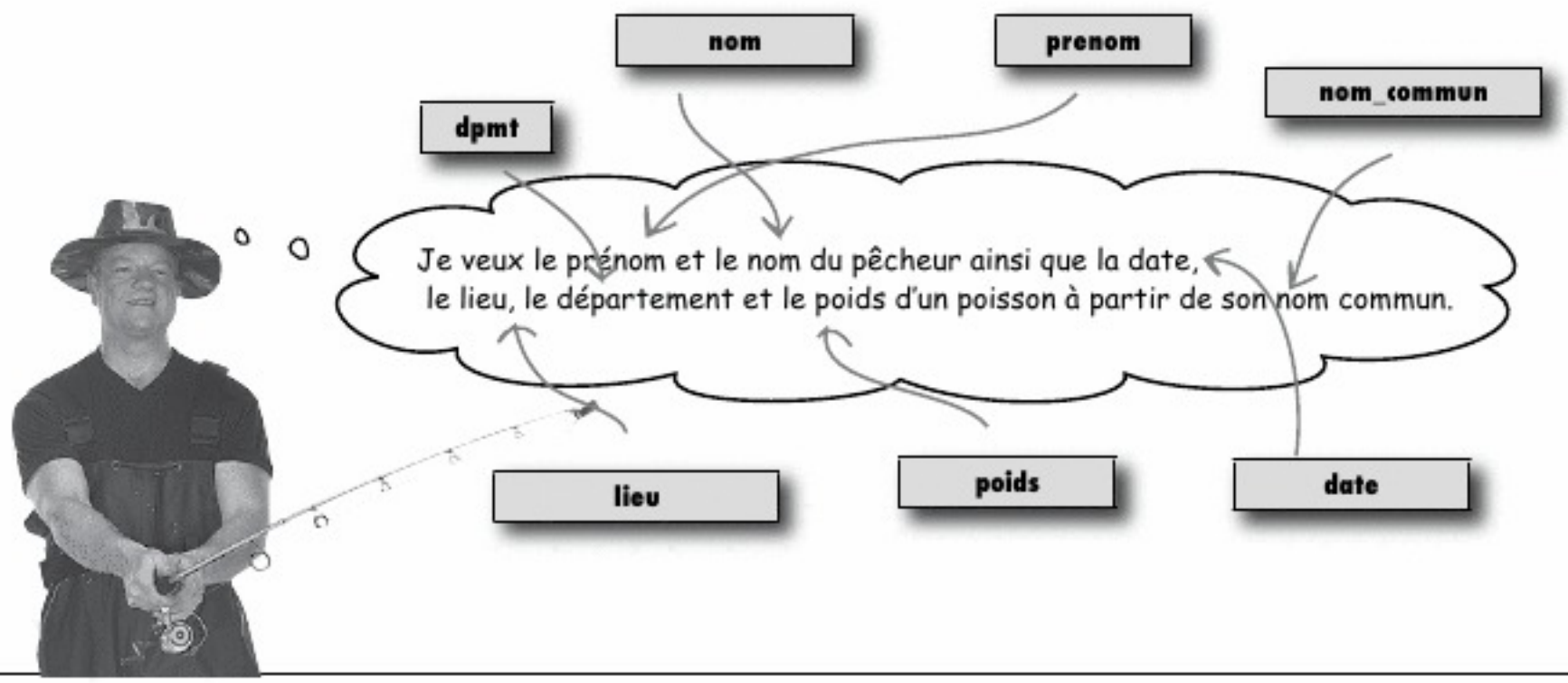


À votre tour. Écrivez une phrase pour Jacques, le journaliste du magazine *Tourne le moulinet*, qui utilise sa table pour sélectionner des détails pour ses articles. Dessinez ensuite des flèches de chaque colonne jusqu'à l'endroit où elle est mentionnée dans la phrase.



Pouvez-vous repérer dans cette phrase les colonnes qu'a utilisé Marc l'ichthyologue pour décrire la façon dont il veut interroger sa table ? Complétez avec les noms de colonnes.

À votre tour. Écrivez une phrase pour Jacques, le journaliste du magazine *Tourne le moulinet*, qui utilise sa table pour séléctionner des détails pour ses articles. Dessinez ensuite des flèches de chaque colonne jusqu'à l'endroit où elle est mentionnée dans la phrase.



On le pourrait mais nous n'avons pas besoin que les données soient divisées à ce point.

Du moins, pas dans ce cas. Si Jacques écrivait un article sur les meilleurs endroits pour attraper un gros poisson en vacances, *alors* il aurait pu vouloir le numéro et le nom de la rue pour que les lecteurs puissent trouver un logement dans les environs.

Mais Jacques n'avait besoin que du lieu et du département, il n'a donc ajouté que le nombre de colonnes nécessaire afin d'économiser de l'espace dans sa base de données. À ce moment-là, il a décidé que ses données étaient assez divisées : elles sont ***atomiques***.

MUSCLEZ VOS NEURONES

À votre avis, que signifie le mot ***atomique*** en termes de données SQL ?

Données atomiques

Qu'est-ce qu'un atome ? Une petite information qui ne peut pas ou ne devrait pas être divisée. C'est la même chose pour vos données. Quand elles sont **ATOMIQUES**, cela signifie qu'elles ont été divisées en **leurs plus petites parties** ***qui ne peuvent pas ou ne doivent pas être divisées.***

En 30 minutes maximum

Prenez un livreur de pizza. Pour arriver là où il va, il n'a besoin que d'un numéro de rue et une adresse dans une seule colonne. Pour son objectif, c'est atomique. Il n'a jamais besoin de chercher un numéro de rue tout seul.

En fait, si ses données étaient divisées en numéro de rue et nom de rue, ses requêtes devraient être plus longues et plus complexes, retardant la livraison de la pizza à votre porte.

Pour le livreur de pizzas, l'adresse tout entière dans une colonne est assez atomique.

```
Fichier Edition Fenêtre Aide SimpleUsineDePizzas
+--------------+-----------------------------+
| commande_no  | adresse                     |
+--------------+-----------------------------+
| 246          | 59 Rapides d'Ajax           |
| 247          | 849 Rue du SQL              |
| 248          | 2348 Place du PMP           |
| 249          | 1978 Mont HTML              |
| 250          | 24 Sources de la Servlet    |
| 251          | 807 Cercle infini           |
| 252          | 32 Place des Design Patterns |
| 253          | 9208 JavaRanch              |
| 254          | 4653 Lotissements EJB       |
| 255          | 8678 Allée de l'ACOO        |
+--------------+-----------------------------+
> SELECT adresse FROM livraisons_pizzas WHERE commande_no = 252;
+-----------------------------+
| adresse                     |
+-----------------------------+
| 32 Place des Design Patterns |
+-----------------------------+
1 row in set (0.04 sec)
```

Le lieu, le lieu, le lieu

Prenez maintenant un agent immobilier. Il est possible qu'il veuille avoir une colonne à part pour le numéro de la rue. Il peut vouloir envoyer une requête sur une rue pour voir toutes les maisons à vendre par numéro de rue. Pour lui, le numéro de rue et le nom de la rue sont chacun atomiques.

Mais pour l'agent immobilier, séparer la rue du numéro de la rue permet de voir toutes les maisons à vendre dans une rue donnée avec une simple requête.

Fichier Edition Fenêtre Aide JeVeuxMaCommission

```
+---------------+---------------------------+------------------+---------+
| numero_rue    | nom_rue                   | type_propriete   | prix    |
+---------------+---------------------------+------------------+---------+
| 59            | Rapides d'Ajax            | local commercial | 189000  |
| 849           | Rue du SQL                | appartement      | 109000  |
| 2348          | Place du PMP              | maison           | 355000  |
| 1978          | Mont HTML                 | appartement      | 134000  |
| 24            | Sources de la Servlet     | maison           | 355000  |
| 807           | Cercle infini             | garage           | 143900  |
| 32            | Place des Design Patterns | maison           | 465000  |
| 9208          | JavaRanch                 | villa            | 699000  |
| 4653          | Rue du SQL                | appartement      | 115000  |
| 8678          | Allée de l'ACOO           | maison           | 355000  |
+---------------+---------------------------+------------------+---------+
> SELECT prix, type_propriete FROM agence_immo WHERE nom_rue = 'Rue du SQL';
+----------+---------------+
| prix     | type_propriete |
+----------+---------------+
| 109000.00 | appartement    |
| 115000.00 | appartement    |
+----------+---------------+
2 rows in set (0.01 sec)
```

Les données atomiques et vos tables

Vous pouvez vous poser certaines questions pour vous aider à trouver ce qu'il vous faut mettre dans vos tables :

1. Quelle est la seule chose que décrit votre table ?

Votre table décrit-elle des clowns, des vaches, des crêpes, des gens ?

2. Comment allez vous utiliser la table pour trouver cette chose ?

Concevez votre table pour qu'elle soit facile à interroger !

3. Vos colonnes contiennent-elles des données atomiques pour que vos requêtes soient courtes et concises ?

il n'y a pas de Questions stupides

Q: Mais les atomes ne sont-ils pas minuscules ? Ne dois-je pas diviser mes données en morceaux *vraiment* minuscules ?

R: Non. Rendre vos données atomiques signifie les diviser pour obtenir les plus petits morceaux dont vous avez besoin pour créer une table efficace et non pas les plus petits morceaux possibles.

Ne divisez pas vos données plus que nécessaire. Si vous n'avez pas besoin de colonnes supplémentaires, ne les ajoutez pas juste pour le plaisir.

Q: En quoi les données atomiques peuvent-elles m'aider ?

R: Elles contribuent à vous assurer de l'exactitude des données de votre table. Par exemple, si vous avez une colonne pour les numéros de rue, vous pouvez faire en sorte que seuls des nombres arrivent dans cette colonne.

Les données atomiques vous permettent également des requêtes plus efficaces parce qu'elles sont plus faciles à écrire et mettent moins de temps à s'exécuter, ce qui s'additionne quand vous avez une grande quantité de données à stocker.

À vos crayons !

Voici les règles officielles des données atomiques. Pour chaque règle, dessinez **deux** tables hypothétiques qui enfreignent la règle.

RÈGLE 1 : Une colonne avec des données atomiques ne peut pas contenir plusieurs valeurs du même genre de données.

La colonne centres_interet de la table de Greg mes_contacts enfreint cette règle.

RÈGLE 2 : Une table avec des données atomiques ne peut pas avoir plusieurs colonnes avec le même genre de données.

La table boissons_faciles enfreint cette règle.

À vos crayons ! Solution

Voici les règles officielles des données atomiques. Pour chaque règle, dessinez **deux** tables hypothétiques qui enfreignent la règle.

RÈGLE 1 : Une colonne avec des données atomiques ne peut pas contenir plusieurs valeurs du même genre de données.

Bien sûr, vos réponses seront différentes mais voici un exemple :

Vous souvenez-vous de la table de Greg ? Elle a une colonne pour les passe-temps qui contient souvent plusieurs centres d'intérêt, rendant les recherches cauchemardesques !

nom_aliment	ingredients
brioche	farine, lait, œuf, levure, huile
salade	laitue, tomate, concombre

C'est pareil ici : imaginez que vous essayiez de trouver une tomate au milieu de tous ces ingrédients.

RÈGLE 2 : Une table avec des données atomiques ne peut pas avoir plusieurs colonnes avec le même genre de données.

Trop de colonnes étudiant !

professeur	etudiant1	etudiant2	etudiant3
Mme Martini	Joël	Romain	Catherine
M. Rocard	Sanjaya	Thomas	Julie

Maintenant que vous connaissez les règles officielles et les trois étapes pour rendre des données atomiques, jetez un coup d'œil à chacune des tables que nous avons déjà vues dans ce livre et expliquez pourquoi elle est ou n'est pas atomique.

Table de Greg, page 47 .....

Table d'évaluation des crêpes, page 78 .....

Table des clowns, page 121 .....

Table des boissons, page 59 .....

Infos sur les poissons, page 160 .....

Des raisons d'être normal

Quand les consultations de vos données décolleront et que vous aurez besoin d'engager plus de concepteurs de bases de données SQL, ne serait-ce pas merveilleux de ne pas avoir à passer des heures à expliquer comment fonctionnent vos tables ?

Hé bien, rendre vos tables NORMALES signifie qu'elles suivent certaines règles standard que vos nouveaux concepteurs comprendront. La bonne nouvelle, c'est que nos tables aux données atomiques sont à mi-chemin de la normalité.

Rendre vos données atomiques est la première étape pour créer une table NORMALE.

Solutions des exercices

Maintenant que vous connaissez les règles officielles et les trois étapes pour rendre des données atomiques, jetez un coup d'œil à chacune des tables que nous avons déjà vues dans ce livre et expliquez pourquoi elle est ou n'est pas atomique.

Table de Greg, page 47 Pas atomique. Les colonnes « centres_interet » et « cherche » enfreignent la règle 1.

Table d'évaluation des crêpes, page 78 Atomique. Contrairement à la table boissons_faciles, chaque colonne contient un type d'information différent et, contrairement à la colonne « activites » de la table des clowns, chaque colonne ne contient qu'une seule information.

Table des clowns, page 121 Pas atomique. La colonne « activites » a plus d'une activité dans certains enregistrements et enfreint donc la règle 1.

Table des boissons, page 59 Pas atomique. Il y a plus d'une colonne « ingredient », ce qui enfreint la règle 2.

Infos sur les poissons, page 160 Atomique. Chaque colonne contient un genre d'information différent et ne contient qu'une seule information.

Les avantages des tables normales

1. Les tables normales n'ont pas de données dupliquées, ce qui réduit la taille de votre base de données.

2. Avec moins de données à parcourir, vos requêtes sont plus rapides.

Parce que même si vos tables sont toutes petites, cela s'additionne.

De plus, les tables grandissent. Si vous commencez avec une table normalisée, vous n'aurez pas besoin de revenir en arrière pour la changer quand vos requêtes seront trop lentes.

Les clowns ne sont pas normaux

Vous vous souvenez de la table des clowns ? Suivre des clowns est devenu une lubie nationale et notre vieille table ne va pas suffire parce que les colonnes `description` et `activites` contiennent *trop* de données. Pour nos objectifs, cette table n'est pas atomique.

Ces deux colonnes sont vraiment difficiles à interroger parce qu'elles contiennent trop de données !

info_clowns

nom	vu_en_dernier	description	activites
Elsie	Maison de retraite La colline des cerises	F, cheveux rouges, robe verte, pieds immenses	ballons, petite voiture
Cornichon	Fête de Max Leblanc	H, cheveux oranges, costume bleu, pieds immenses	mime
Mme Câlin	La boutique du ballon	F, chemise jaune, pantalon bleu trop grand	klaxon, parapluie
Bobo	Fête d'Éric Gris	H, cigare, cheveux noirs, chapeau minuscule	violon
Clarabelle	Maison de retraite Belmont	F, cheveux roses, fleur énorme, robe bleue	cris, danse
Scooter	Hôpital Les Chênes	H, cheveux bleus, costume rouge, gros nez	ballons
Zippo	Galerie Pierre du moulin	F, costume orange, pantalon trop grand	chant
Poulette	Les autos de Gilou	F, tout en rose et strass	balançoire, petite voiture
Bozo	Parc Gicard Destin	H, travesti, robe à pois	chant, danse
Sniff	Chez Valérie	H, costume vert et violet, nez pointu	Monte dans une petite voiture

À vos crayons !

Rendons la table des clowns plus atomique. En supposant que vous ayez besoin de lancer des recherches sur les données des colonnes `description` et `activites`, ainsi que `vu_en_dernier`, notez de meilleurs choix pour les colonnes.

Réponses page 195.

À mi-chemin de l'1FN

Rappelez-vous, votre table n'est qu'à mi-chemin de la normalité quand elle contient des données atomiques. Quand nous serons complètement normaux nous serons à la PREMIÈRE FORME NORMALE (*first normal form*) ou 1FN.

Pour être à la 1FN, une table doit suivre ces deux règles :

Nous savons déjà comment faire cela.

Chaque ligne de données doit contenir des valeurs atomiques.

Pour rendre vos tables complètement normales, nous devons donner à chaque enregistrement une Clé Primaire.

Chaque ligne de données doit avoir un identifiant unique, appelé une Clé Primaire.

MUSCLEZ VOS NEURONES

Quels types de colonnes feraient de bonnes Clés Primaires, à votre avis ?

Les règles de la CLÉ PRIMAIRE

La colonne de votre table qui sera votre clé primaire doit être désignée comme telle quand vous créez la table. Dans quelques pages, nous allons créer une table et désigner une clé primaire, mais avant cela, regardons de plus près ce qu'est une clé primaire.

Une clé primaire est une colonne de votre table qui rend chaque enregistrement unique.

On utilise la clé primaire pour identifier de façon unique chaque enregistrement

Ce qui signifie que les données de la colonne de la clé primaire ne peuvent pas être répétées. Prenez une table avec les colonnes ci-dessous. Lesquelles feraient de bonnes clés primaires, à votre avis ?

NSS	nom	prenom	numero_telephone

Comme les Numéros de Sécurité Sociale sont assignés à chacun de façon unique, ils pourraient être une clé primaire.

Ces trois colonnes peuvent toutes contenir des valeurs dupliquées ; par exemple, vous aurez sans doute un enregistrement pour plus d'une personne appelée Nicolas ou plusieurs personnes qui vivent ensemble et partagent un numéro de téléphone, donc, ce ne sont sans doute pas de bons choix pour la clé primaire.

Ouvrez les yeux !

Attention si vous utilisez des NSS comme Clés Primaires pour vos enregistrements.

Avec l'augmentation des vols d'identité, les gens ne veulent plus donner leur NSS et avec raison. Ils sont trop importants pour prendre des risques. Pouvez-vous absolument garantir que votre base de données est sûre ? Si elle ne l'est pas, tous ces NSS peuvent être volés et avec eux les identités de vos clients.

Une clé primaire ne peut pas être NULL

Si elle est Null, elle ne peut pas être unique puisque d'autres enregistrements peuvent aussi être NULL.

La clé primaire doit recevoir une valeur quand l'enregistrement est inséré

Si vous insérez un enregistrement sans clé primaire, vous courez le risque de finir avec une clé primaire NULL et de dupliquer des lignes de votre table, ce qui enfreint la première forme normale.

La clé primaire doit être compacte

Une clé primaire ne doit contenir que les informations nécéssaires pour être unique et rien de plus.

Les valeurs de la clé primaire ne peuvent pas être modifiées

Si vous pouviez modifier la valeur de votre clé, vous risqueriez de la définir accidentellement à une valeur que vous avez déjà utilisée. Rappelez-vous, elle doit rester unique.

MUSCLEZ VOS NEURONES

Etant données toutes ces règles, pouvez-vous trouver une bonne clé primaire à utiliser dans une table ?

Retournez voir les tables du livre. Y a-t-il des colonnes qui contiennent des valeurs réellement uniques ?

Attendez, alors si je ne peux pas utiliser le NSS comme clé primaire mais que la clé doit quand même être compacte, pas NULL et inchangeable, que puis-je utiliser ?

Il est possible que la meilleure clé primaire soit une *nouvelle* clé primaire.

Pour créer des clés primaires, votre meilleure option peut être de créer une colonne qui contienne un nombre unique. Pensez à une table avec les infos des gens mais aussi une colonne supplémentaire contenant un nombre. Dans l'exemple ci-dessous, appelons-la `ID`.

Sans la colonne `ID`, les enregistrement pour Nicolas Dupré seraient identiques. Mais dans ce cas il s'agit bien de deux personnes différentes. La colonne `ID` rend ces enregistrements uniques. Cette table répond à la première forme normale.

id	nom	prenom	surnom
1	Dupré	Nicolas	Nico
2	Pignon	François	Panpan
3	Dupré	Nicolas	Nico
4	Petrillo	Marie	Maria
5	Franken	Emeline	Em

Un enregistrement de Nicolas Dupré.

Un autre enregistrement pour Nicolas Dupré, mais la colonne ID montre que c'est un enregistrement unique, c'est donc un Nicolas Dupré différent du premier.

Pour les passionnés

Il y a un grand débat dans le monde du SQL sur l'utilisation de clés primaires ***synthétiques***, ou artificielles (comme la colonne ID ci-dessus) contre l'utilisation de clés ***naturelles*** : des données déjà présentes dans la table (comme un numéro VIN pour une voiture ou un numéro de sécu). Nous ne prenons pas parti, mais nous aborderons plus en détails les clés primaires dans le chapitre 7.

il n'y a pas de
Questions stupides

Q: Vous dites « première » forme normale. Cela signifie-t-il qu'il existe une deuxième forme normale ? Ou une troisième ?

R: Oui, il existe en fait une deuxième et une troisième formes normales, chacune ahhérant à des ensembles de règles de plus en plus rigoureux. Nous verrons la deuxième et la troisième formes normales dans le chapitre 7.

Q: Nous avons donc modifié nos tables pour qu'elles aient des données atomiques. Certaines sont-elles déjà en 1FN ?

R: Non. Jusqu'ici, aucune des tables que nous avons créées n'a de clé primaire, de valeur unique.

Q: La colonne commentaires dans la table des crêpes ne me semble vraiment pas atomique. Je veux dire, il n'y a aucun moyen pratique d'interroger cette colonne facilement.

R: Vous avez tout à fait raison. Ce champ n'est pas particulièrement atomique mais, à ce moment-là, notre conception de la table ne demandait pas qu'il le soit. Si nous voulions restreindre les commentaires à un ensemble de mots spécifiques et prédéterminés, ce champ pourrait être atomique. Mais dans ce cas il ne contiendrait pas de commentaires vrais et spontanés.

Vers le NORMAL

Il est temps de faire un pas en arrière pour normaliser nos tables. Nous devons rendre nos données atomiques et ajouter des clés primaires. Créer une clé primaire se fait normalement quand on écrit le code `CREATE TABLE`.

Vous souvenez-vous de la façon d'ajouter des colonnes à une table existante ?

Réparer la table de Greg

D'après tout ce que vous avez vu jusqu'ici, voici comment vous devez réparer la table de Greg :

Etape 1 pour réparer la table de Greg : SELECTionner toutes vos données et les enregistrer quelque part.

Etape 2 pour réparer la table de Greg : Créer une nouvelle table normale.

Etape 3 pour réparer la table de Greg : INSERTion de toutes ces vieilles données dans la nouvelle table en modifiant chaque ligne pour qu'elle corresponde à la nouvelle structure de la table.

Donc maintenant vous pouvez jeter votre vieille table avec un DROP.

Alors. Nous savons que la table de Greg n'est pas parfaite.

Elle n'est pas atomique et n'a pas de clé primaire. Mais heureusement pour Greg, vous n'avez ***pas*** besoin de garder la vieille table et vous **n'avez pas besoin de jeter vos données**.

Nous pouvons ajouter une clé primaire à la table de Greg et rendre les colonnes plus atomiques simplement en utilisant une nouvelle commande. Mais d'abord, embarquons pour un petit voyage dans le passé…

Le CREATE TABLE que nous avons écrit

Greg a besoin d'une clé primaire et après toute cette discussion sur les données atomiques, il réalise qu'il pourrait faire plusieurs choses pour rendre ses colonnes plus atomiques. Avant de regarder comment réparer la table existante, voyons comment nous aurions pu créer la table en premier lieu !

Voici la table que nous avons créée tout au début, dans le premier chapitre.

```
CREATE TABLE mes_contacts
(
    nom VARCHAR(30),
    prenom VARCHAR(20),
    email VARCHAR(50),
    sexe CHAR(1),
    anniversaire DATE,
    profession VARCHAR(50),
    lieu VARCHAR(50),
    statut VARCHAR(20),
    centres_interet VARCHAR(100),
    cherche VARCHAR(100)
);
```

Pas de clé primaire ici

Ces colonnes auraient-elles pu être plus atomiques quand nous avons défini la table ?

MUSCLEZ VOS NEURONES

Que se passe-t-il si vous n'avez pas votre vieux `CREATE TABLE` imprimé quelque part ? Pouvez-vous trouver un moyen d'obtenir le code ?

Montre-moi l'~~argent~~ la table

Et si vous utilisiez la commande `DESCRIBE mes_contacts` pour voir le code que vous avez utilisé quand vous avez défini la table ? Vous verrez quelque chose qui ressemble à cela :

```
Fichier Edition Fenêtre Aide EncoreLaListeDeGreg
+----------------+-------------+------+-----+---------+------------+
| Column         | Type        | Null | Key | Default | Extra      |
+----------------+-------------+------+-----+---------+------------+
| nom            | varchar(30) | YES  |     | NULL    |            |
| prenom         | varchar(20) | YES  |     | NULL    |            |
| email          | varchar(50) | YES  |     | NULL    |            |
| sexe           | char(1)     | YES  |     | NULL    |            |
| anniversaire   | date        | YES  |     | NULL    |            |
| profession     | varchar(50) | YES  |     | NULL    |            |
| lieu           | varchar(50) | YES  |     | NULL    |            |
| statut         | varchar(20) | YES  |     | NULL    |            |
| centres_interet | varchar(100) | YES  |     | NULL    |            |
| cherche        | varchar(100) | YES  |     | NULL    |            |
+----------------+-------------+------+-----+---------+------------+
```

Mais ce que nous voulons c'est voir le code **CREATE**, pas les champs de la table, pour pouvoir trouver ce que nous aurions dû faire au tout début sans devoir réécrire l'instruction `CREATE`.

L'instruction `SHOW CREATE TABLE` va renvoyer une instruction `CREATE TABLE` qui peut recréer exactement notre table, sans aucune donnée à l'intérieur. De cette façon, vous pouvez toujours voir comment la table que vous regardez peut être créée. Essayez-la :

```
SHOW CREATE TABLE mes_contacts;
```

Une commande pour gagner du temps

Regardez le code que nous avons utilisé pour créer la table page 183 et le code ci-dessous que vous donne le `SHOW CREATE TABLE mes_contacts`. Ils ne sont pas identiques, mais si vous collez le code ci-dessous dans une commande `CREATE TABLE`, le résultat final sera le même. Vous n'avez pas besoin d'enlever les guillemets obliques ou les paramètres des données, c'est simplement plus clair si vous le faites.

Les marques autour du nom de la table et des noms de colonnes s'appellent des guillemets obliques. Ils apparaissent quand on exécute la commande SHOW CREATE TABLE.

À moins qu'on ne dise le contraire au logiciel SQL, il assume que toutes les valeurs sont NULL par défaut.

Il vaut mieux préciser si une colonne peut contenir des NULL ou pas quand nous créons notre table.

```
CREATE TABLE `mes_contacts`
(
    `nom` varchar(30) default NULL,
    `prenom` varchar(20) default NULL,
    `email` varchar(50) default NULL,
    `sexe` char(1) default NULL,
    `anniversaire` date default NULL,
    `profession` varchar(50) default NULL,
    `lieu` varchar(50) default NULL,
    `statut` varchar(20) default NULL,
    `centres_interet` varchar(100) default NULL,
    `cherche` varchar(100) default NULL
) ENGINE=MyISAM DEFAULT CHARSET=latin1
```

Ne vous préoccupez pas de la dernière ligne de texte après la parenthèse fermante. Elle précise la façon dont les données seront conservées et l'ensemble de caractères à utiliser. Les paramètres par défaut font l'affaire pour l'instant.

À moins d'avoir supprimé la table d'origine, vous devrez donner un nouveau nom à celle-ci.

Bien que vous puissiez rendre le code plus clair (en enlevant la dernière ligne et les guillemets obliques), vous pouvez simplement le copier-coller pour créer une table.

CREATE TABLE avec une CLE PRIMAIRE

Voici le code que notre `SHOW CREATE TABLE mes_contacts` nous a donné. Nous avons enlevé les guillemets obliques et la dernière ligne. Nous avons ajouté une colonne `id_contact` en haut de la liste des colonnes, que nous définissons comme `NOT NULL` et en bas de la liste, nous avons ajouté une ligne `PRIMARY KEY`, dans laquelle nous définissons notre nouvelle colonne `id_contact` comme étant la clé primaire.

Rappelez-vous, la colonne de la clé primaire doit être NOT NULL ! Si la clé primaire contient une valeur NULL ou pas de valeur, vous ne pouvez pas être certain qu'elle identifiera chaque ligne de la table de façon unique.

```
CREATE TABLE mes_contacts
(
  id_contact INT NOT NULL,
  nom varchar(30) default NULL,
  prenom varchar(20) default NULL,
  email varchar(50) default NULL,
  sexe char(1) default NULL,
  anniversaire date default NULL,
  profession varchar(50) default NULL,
  lieu varchar(50) default NULL,
  statut varchar(20) default NULL,
  centres_interet varchar(100) default NULL,
  cherche varchar(100) default NULL,
  PRIMARY KEY (id_contact)
);
```

Nous avons créé une nouvelle colonne appelée id_contact qui contient une valeur entière (integer), la clé primaire de notre table. Chaque valeur de cette colonne est unique et rend notre table atomique.

C'est là que nous spécifions où est la clé primaire. La syntaxe est assez simple : nous mettons juste PRIMARY KEY et, entre parenthèses, le nom de la colonne que nous utilisons pour la clé primaire. Ici, notre nouvelle colonne id_contact.

il n'y a pas de
Questions stupides

Q: Vous dites donc que la CLÉ PRIMAIRE ne peut pas être NULL. Y a-t-il autre chose qui l'empêche d'être dupliquée ?

R: Vous, tout simplement. Quand vous utilisez `INSERT` pour insérer des valeurs dans votre table, vous insérez à chaque fois une nouvelle valeur dans la colonne `id_contact`. Par exemple, la première instruction `INSERT` va définir `id_contact` comme `1`, l'`id_contact` suivant sera `2`, etc.

Q: C'est assez pénible de devoir assigner une nouvelle valeur à cette colonne CLÉ PRIMAIRE à chaque fois que j'insère un nouvel enregistrement. Existe-t-il un moyen plus simple ?

R: Il en existe deux. L'un est d'utiliser comme clé primaire une colonne de vos données qui est unique. Nous avons mentionné que cela peut poser des problèmes (par exemple, le problème avec l'utilisation des numéros de sécurité sociale).

Le moyen le plus simple est de créer une toute nouvelle colonne pour garder une valeur unique, comme `id_contact` sur la page ci-contre. Vous pouvez dire à votre logiciel SQL de le remplir automatiquement avec un nombre en utilisant des mots-clés. Tournez la page pour plus de détails.

Q: Puis-je utiliser SHOW pour autre chose que la commande CREATE ?

R: Vous pouvez utiliser SHOW pour afficher les colonnes de votre table :

```
SHOW COLUMNS FROM nomtable;
```

Cette commande affiche toutes les colonnes de votre table et leur type de données ainsi que tous les autres détails spécifiques aux colonnes.

```
SHOW CREATE DATABASE nombdd;
```

Tout comme SHOW CREATE TABLE, vous obtenez la commande qui peut recréer votre base de données à l'identique.

```
SHOW INDEX FROM nomtable;
```

Cette commande affiche les colonnes qui sont indexées et leur type d'index. Jusqu'ici, le seul index que nous avons vu est la clé primaire, mais cette commande deviendra plus utile quand vous en saurez plus.

Il y a encore une commande TRÈS utile :

```
SHOW WARNINGS;
```

Si vous obtenez un message sur votre console disant que votre commande SQL a provoqué des avertissements, saisissez ceci pour les voir.

Il y en a encore d'autres mais ces commandes sont celles qui ont un rapport avec ce que nous avons fait jusqu'ici.

Q: Alors qu'est-ce que c'est que ce guillemet oblique qui apparaît quand j'utilise un SHOW CREATE TABLE ? Êtes-vous sûr que je n'en ai pas besoin ?

R: Il existe parce que parfois votre SGBDR peut ne pas être capable de déterminer si un nom de colonne est un nom de colonne. Si vous utilisez les guillemets obliques autour de vos noms de colonnes, vous pouvez (même si c'est une très mauvaise idée) utiliser un mot-clé réservé SQL comme nom de colonne.

Par exemple, supposez que, pour une raison bizarre, vous vouliez appeler une colonne `select`. Cette déclaration de colonne ne fonctionnerait pas :

```
select varchar(50)
```

Mais celle-ci fonctionnerait :

```
`select` varchar(50)
```

Q: Pourquoi ne pas utiliser des mots-clés comme noms de colonnes alors ?

R: Vous pouvez le faire mais c'est une mauvaise idée. Imaginez comme vos requêtes deviendraient confuses et en plus vous devrez taper ces guillemets obliques alors que vous pouvez très bien ne pas les utiliser. De plus, `select` n'est pas un très bon nom de colonne, il ne vous dit rien sur les données qui s'y trouvent.

Q: Où puis-je trouver le guillemet oblique sur mon clavier ?

R: Il se trouve sous la touche 7. Vous l'obtenez en tapant Alt Gr + 7.

1, 2, 3... auto incrémentation

Ajouter le mot-clé `AUTO_INCREMENT` à notre colonne `id_contact` permet à notre logiciel SQL de remplir cette colonne automatiquement avec une valeur qui commence par 1 à la ligne 1 et augmente par incrémentation d'1.

```
CREATE TABLE mes_contacts
(
    id_contact INT NOT NULL AUTO_INCREMENT,
    nom varchar(30) default NULL,
    prenom varchar(20) default NULL,
    email varchar(50) default NULL,
    sexe char(1) default NULL,
    anniversaire date default NULL,
    profession varchar(50) default NULL,
    lieu varchar(50) default NULL,
    statut varchar(20) default NULL,
    centres_interet varchar(100) default NULL,
    cherche varchar(100) default NULL,
    PRIMARY KEY (id_contact)
);
```

C'est tout. Avec la plupart des versions de SQL, ajoutez simplement le mot-clé AUTO_INCREMENT (que les utilisateurs de MS SQL fassent attention, le mot-clé est INDEX accompagné d'une valeur de départ et d'une valeur d'incrémentation. Voyez votre référence MS SQL pour des infos plus précises).

Le mot-clé fait grosso modo ce que vous pouvez en attendre : il commence à 1 et augmente de 1 à chaque fois que vous insérez une nouvelle ligne.

Que se passerait-il à votre avis ?

Mieux encore, essayez-le vous-même et voyez ce qui se passe.

1 Écrivez ci-dessous une instruction CREATE TABLE pour stocker les prénoms et noms des gens. Votre table doit avoir une colonne clé primaire avec AUTO_INCREMENT et deux autres colonnes atomiques.

. .....

. .....

. .....

2 Ouvrez votre terminal SQL ou votre interface graphique et lancez votre instruction CREATE TABLE.

3 Essayez chacune des instructions INSERT ci-dessous. Entourez celles qui fonctionnent.

```
INSERT INTO votre_table (id, prenom, nom)
VALUES (NULL, 'Marcelle', 'Lemaire');

INSERT INTO votre_table (id, prenom, nom)
VALUES (1, 'Jean', 'Lemaire');

INSERT INTO votre_table
VALUES ('', 'Joseph', 'Lemaire');

INSERT INTO votre_table (prenom, nom)
VALUES ('Sandra', 'Lemaire');

INSERT INTO votre_table (id, prenom, nom)
VALUES (99, 'Pierre', 'Lemaire');
```

4 Tous les Lemaire sont-ils passés ? Dessinez votre table et son contenu après avoir essayé les instructions INSERT.

votre_table

id	prenom	nom

Solutions des exercices

1 Écrivez ci-dessous une instruction CREATE TABLE pour stocker les prénoms et noms des gens. Votre table doit avoir une colonne clé primaire avec AUTO_INCREMENT et deux autres colonnes atomiques.

```
CREATE TABLE votre_table
(
id INT NOT NULL AUTO_INCREMENT,
prenom VARCHAR(20),
nom VARCHAR(30),
PRIMARY KEY (id)
);
```

2 Ouvrez votre terminal SQL ou votre interface graphique et lancez votre instruction CREATE TABLE.

3 Essayez chacune des instructions INSERT ci-dessous. Entourez celles qui fonctionnent.

```
INSERT INTO votre_table (id, prenom, nom)
VALUES (NULL, 'Marcelle', 'Lemaire');

INSERT INTO votre_table (id, prenom, nom)
VALUES (1, 'Jean', 'Lemaire');

INSERT INTO votre_table
VALUES ('', 'Joseph', 'Lemaire');

INSERT INTO votre_table (prenom, nom)
VALUES ('Sandra', 'Lemaire');

INSERT INTO votre_table (id, prenom, nom)
VALUES (99, 'Pierre', 'Lemaire');
```

Cela fonctionne peut-être avec votre SGBDR. Mais MySQL ne vous autorise pas à saisir " pour un numéro auto_incrémenté. Si vous ne voulez pas saisir de numéro pour votre colonne auto-incrémentée, vous pouvez calquer votre instruction sur celle de Sandra.

Cette dernière instruction « fonctionne », mais réécrit sur la valeur de la colonne AUTO_INCREMENTée.

4 Tous les Lemaire sont-ils passés ? Dessinez votre table et son contenu après avoir essayé les instructions INSERT.

votre_table

id	prenom	nom
1	Marcelle	Lemaire
2	Sandra	Lemaire
99	Pierre	Lemaire

On dirait que nous avons perdu Jean parce que nous essayions de lui donner un index qui était déjà assigné à Marcelle. Marcelle, Marcelle, Marcelle !

il n'y a pas de
Questions stupides

Q: Pourquoi la première requête, celle avec NULL pour la colonne `id`, a inséré la ligne alors que `id` est NOT NULL ?

R: Même si cela ne semble pas devoir marcher, l'AUTO_INCREMENT ignore tout simplement le NULL. Cependant, si ce n'était pas AUTO_INCREMENTé, vous obtiendriez une erreur et la ligne ne serait pas insérée. Essayez.

Écoutez, vous ne me rassurez pas. Bien sûr, je peux coller le code de SHOW CREATE TABLE, mais j'ai toujours l'impression que je vais devoir envoyer un drop dans ma table et recommencer à entrer tous ces enregistrements simplement pour ajouter la colonne clé primaire après-coup.

Vous ne devrez pas tout recommencer. Au lieu de cela, vous pouvez utiliser une instruction ALTER.

Une table contenant des données n'a pas besoin d'être jetée puis recréée. En fait, nous pouvons modifier une table existante. Mais pour ce faire, nous allons emprunter au chapitre 5 l'instruction `ALTER` et certains de ses mots-clés.

Ajouter une CLE PRIMAIRE à une table existante

Voici le code pour ajouter une clé primaire avec `AUTO_INCREMENT` à la table `mes_contacts` de Greg. (C'est une longue commande, vous aurez donc besoin de tourner votre livre.)

Voici notre nouvelle commande SQL, ALTER.

Voici le code pour ajouter la nouvelle colonne à la table. Cela vous semble familier ?

FIRST dit au logiciel de placer la nouvelle colonne en premier sur la liste. C'est optionnel mais c'est une bonne habitude de placer votre clé primaire en premier.

```
ALTER TABLE mes_contacts
ADD COLUMN id_contact INT NOT NULL AUTO_INCREMENT FIRST,
ADD PRIMARY KEY (id_contact);
```

ADD COLUMN dit d'ajouter une colonne à la table et de l'appeler id_contact.

Vous devez reconnaître la ligne qui désigne la clé primaire.

MUSCLEZ VOS NEURONES

Pensez-vous que cela ajoutera des valeurs à la nouvelle colonne `id_contact` pour les enregistrements déjà présents dans la table ou seulement pour les enregistrements à venir ? Comment pouvez-vous le vérifier ?

ALTER TABLE et l'ajout d'une CLE PRIMAIRE

Essayez le code vous-même. Ouvrez votre terminal SQL. Avec USE, entrez dans la base de données `liste_greg` et saisissez cette commande :

```
Fichier Edition Fenêtre Aide Altérations
> ALTER TABLE mes_contacts
    -> ADD COLUMN id_contact INT NOT NULL AUTO_INCREMENT FIRST,
    -> ADD PRIMARY KEY (id_contact);

Query OK, 50 rows affected (0.04 sec)
Records: 50  Duplicates: 0  Warnings: 0
```

Ceci nous dit que la colonne a été ajoutée aux 50 enregistrements que nous avons déjà dans notre table. Vous n'en aurez pas autant.

Pour voir ce qui est arrivé à votre table, essayez un
`SELECT * from mes_contacts;`

La colonne id_contact a été ajoutée en premier dans la table, avant toutes les autres colonnes.

```
Fichier Edition Fenêtre Aide Altérations
+------------+------------+------------+---------------------
| id_contact | nom        | prenom     | email
+------------+------------+------------+---------------------
|     1      | Bouquet    | Carole     | carole_bouquet@pizzapre
|     2      | Joffe      | Pierre     | joffe@canardsdesimulat
|     3      | Nouveau    | Amandine   | aman2luv@pizzaprestiss
|     4      | Garcia     | Edouard    | ed99@basenhaut.com
|     5      | Tourne     | Marianne   | mamatourne@pizzapres
|     6      | Cousteau   | Christophe | ccousteau@[illegible]
```

Comme nous avons utilisé AUTO_INCREMENT, la colonne a été complétée et tous les enregistrements de la table ont été mis à jour.

La prochaine fois que nous insérons un nouvel enregistrement, la colonne id_contact recevra une valeur supérieure d'un à la plus haute valeur d'id_contact de la table. Si le dernier enregistrement a un id_contact de 23, le suivant sera 24.

Souvenez-vous, ce n'est pas la fin de la table ; Greg a beaucoup de contacts.

Greg obtiendra-t-il sa colonne de numéros de téléphone ?
Passez au chapitre 5 pour le savoir.

Boîte à outils SQL

Le chapitre 4 est derrière vous. Regardez tous les nouveaux outils que vous avez ajouté à votre boîte maintenant ! Pour une liste exhaustive des outils du livre, voyez l'annexe C.

DONNÉES ATOMIQUES RÈGLE 1:

Les données atomiques ne peuvent pas avoir plusieurs morceaux du même genre de données dans la même colonne.

DONNÉES ATOMIQUES RÈGLE 2:

Les données atomiques ne peuvent pas avoir plusieurs colonnes avec le même genre de données.

DONNÉES ATOMIQUES

Les données de vos colonnes sont atomiques si elles sont divisées en morceaux, les plus petits dont vous ayez besoin.

SHOW CREATE TABLE

Utilisez cette commande pour voir la syntaxe correcte pour créer une table existante.

CLÉ PRIMAIRE

Une colonne ou un ensemble de colonnes qui identifie de façon unique une ligne de données dans une table.

PREMIÈRE FORME NORMALE (1FN)

Chaque ligne de données doit contenir des valeurs atomiques et avoir un identifiant unique.

AUTO_INCREMENT

Si vous l'utilisez dans votre déclaration de colonne, cette colonne recevra automatiquement une valeur entière unique à chaque fois qu'une commande INSERT est exécutée.

Solutions des exercices

Rendons la table des clowns plus atomique. En supposant que vous ayez besoin de lancer des recherches sur les données des colonnes `description` et `activites`, ainsi que `vu_en_dernier`, notez de meilleurs choix pour les colonnes.

Il n'y a pas de réponse correcte précise ici.

Le mieux que vous puissiez faire est de sortir des choses comme le sexe, la couleur de la chemise, la couleur du pantalon, le type de chapeau, l'instrument de musique, le moyen de transport, les ballons (avec oui ou non pour valeurs), le chant (avec oui ou non pour valeurs), la danse (avec oui ou non pour valeurs).

Pour rendre cette table atomique, vous devez diviser ces diverses activités et toutes ces caractéristiques de description et les placer dans des colonnes séparées.

Des points de bonus si vous vouliez séparer la colonne lieu en adresse, ville et département !

Réécrire le passé

Avez-vous déjà souhaité pouvoir corriger les erreurs de votre passé ? Hé bien, voici votre chance. En utilisant la **commande ALTER**, vous pouvez appliquer toutes les leçons que vous avez apprises à des tables que vous avez conçues il y a des jours, des mois ou même des années. Mieux encore, vous pouvez le faire sans affecter vos données. Quand vous en aurez fini ici, vous saurez ce que la **normalité** signifie vraiment et vous serez capable de l'appliquer à vos tables, passées et présentes.

Nous avons besoin de faire quelques changements

Greg veut faire d'autres modifications dans sa table mais il ne veut pas perdre de données.

Fichier Edition Fenêtre Aide InstructionSaisie

```
+------------+------------+------------+-----------------------
| id_contact | nom        | prenom     | email
+------------+------------+------------+-----------------------
|     1      | Bouquet    | Carole     | carole_bouquet@pizzapre
|     2      | Joffe      | Pierre     | joffe@canardsdesimulat
|     3      | Nouveau    | Amandine   | aman2luv@pizzaprestiss
|     4      | Garcia     | Edouard    | ed99@basenhaut.com
|     5      | Tourne     | Marianne   | mamatourne@pizzaprest
|     6      | Cou[illegible]tea[illegible] | [illegible]hris[illegible]he | [illegible]cousteau@[illegible]
```

Oui, vous pouvez facilement utiliser ALTER TABLE pour l'ajouter.

En fait, nous pensons que vous devriez faire un essai vous-même puisque vous avez déjà rencontré la commande ALTER. Faites l'exercice suivant pour obtenir votre code !

À vos crayons !

Regardez de plus près la commande ALTER TABLE que nous avons utilisée pour ajouter la colonne de la clé primaire dans le chapitre 4 et voyez si vous pouvez trouver votre propre commande pour ajouter une colonne telephone qui puisse contenir 10 chiffres. Notez bien que vous n'aurez pas besoin de tous les mots-clés dans votre nouvelle commande.

```
ALTER TABLE mes_contacts
ADD COLUMN id_contact INT NOT NULL AUTO_INCREMENT FIRST,
ADD PRIMARY KEY (id_contact);
```

Écrivez ici votre commande ALTER TABLE :

. .....

. .....

. .....

Vous pouvez même préciser au logiciel où mettre la colonne telephone avec le mot-clé AFTER. Voyez si vous pouvez trouver où placer le mot-clé pour ajouter la nouvelle colonne juste après la colonne prenom.

Écrivez ici votre nouvelle commande ALTER TABLE :

. .....

. .....

. .....

À vos crayons ! Solution

Regardez de plus près la commande ALTER TABLE que nous avons utilisée pour ajouter la colonne de la clé primaire dans le chapitre 4 et voyez si vous pouvez trouver votre propre commande pour ajouter une colonne telephone qui puisse contenir 10 chiffres. Notez bien que vous n'aurez pas besoin de tous les mots-clés dans votre nouvelle commande.

```
ALTER TABLE mes_contacts
ADD COLUMN id_contact INT NOT NULL AUTO_INCREMENT FIRST,
ADD PRIMARY KEY (id_contact);
```

Les mots-clés de l'exemple précédent que nous avons laissés de côté sont NOT NULL, AUTO_INCREMENT, et FIRST.

Écrivez ici votre commande ALTER TABLE :

Le nom de la table que nous altérons est toujours mes_contacts.

```
ALTER TABLE mes_contacts
ADD COLUMN telephone VARCHAR(10);
```

Nous avons assumé que tous nos numéros de téléphone auraient 10 caractères de long. Greg n'a pas pensé aux numéros internationaux.

Voici la partie qui précise à la commande ALTER la façon exacte dont vous voulez modifier la table.

Le nom de la nouvelle colonne est telephone.

Vous pouvez même préciser au logiciel où mettre la colonne telephone avec le mot-clé AFTER. Voyez si vous pouvez trouver où placer le mot-clé pour ajouter la nouvelle colonne juste après la colonne prenom.

Écrivez ici votre nouvelle commande ALTER TABLE :

```
ALTER TABLE mes_contacts
ADD COLUMN telephone VARCHAR(10)
AFTER prenom;
```

Le mot-clé AFTER suivi par le nom de la colonne après laquelle vous voulez placer la nouvelle colonne. Ceci place la colonne telephone juste après la colonne prenom.

AFTER est en option. Si vous ne l'utilisez pas, la colonne est ajoutée à la fin de la table.

Vous avez vu que vous pouvez utiliser les mots-clés *FIRST* et *AFTER votre_colonne*, mais vous pouvez également utiliser *BEFORE votre_colonne* et *LAST*.
Ainsi que *SECOND*,*THIRD*, etc., vous voyez le principe.

En coulisse

Des mots-clés SQL sur le frigo

Utilisez les aimants ci-dessous pour modifier la position de la colonne telephone qui est ajoutée. Créez autant de commandes différentes que vous le pouvez.

telephone	id_contact	nom	prenom	email

```
ALTER TABLE mes_contacts
ADD COLUMN telephone VARCHAR(10)
```

id_contact	nom	prenom	email	telephone

```
ALTER TABLE mes_contacts
ADD COLUMN telephone VARCHAR(10)
```

id_contact	telephone	nom	prenom	email

```
ALTER TABLE mes_contacts
ADD COLUMN telephone VARCHAR(10)
```

id_contact	nom	telephone	prenom	email

```
ALTER TABLE mes_contacts
ADD COLUMN telephone VARCHAR(10)
```

Ajoutez vos aimants à la fin de l'instruction.

FIRST

nom

LAST

SECOND

AFTER

BEFORE

;

FIFTH

Utilisez le point-virgule autant de fois que nécessaire.

Des mots-clés SQL sur le frigo - SOLUTION

Utilisez les aimants ci-dessous pour modifier la position de la colonne telephone qui est ajoutée. Créez autant de commandes différentes que vous le pouvez.

```
ALTER TABLE mes_contacts
ADD COLUMN telephone VARCHAR(10) FIRST ;
```

FIRST place la colonne telephone avant toutes les autres.

telephone	id_contact	nom	prenom	email

```
ALTER TABLE mes_contacts
ADD COLUMN telephone VARCHAR(10) LAST ;

ALTER TABLE mes_contacts
ADD COLUMN telephone VARCHAR(10) FIFTH ;

ALTER TABLE mes_contacts
ADD COLUMN telephone VARCHAR(10) ;
```

LAST place la colonne telephone après toutes les autres colonnes, de même que FIFTH ou ne pas ajouter de position du tout.

id_contact	nom	prenom	email	telephone

```
ALTER TABLE mes_contacts
ADD COLUMN telephone VARCHAR(10) SECOND ;

ALTER TABLE mes_contacts
ADD COLUMN telephone VARCHAR(10)
BEFORE nom ;
```

SECOND place la colonne telephone en second, ce que fait aussi BEFORE (si vous l'utilisez avec la colonne nom).

id_contact	telephone	nom	prenom	email

```
ALTER TABLE mes_contacts
ADD COLUMN telephone VARCHAR(10)
AFTER nom ;
```

AFTER nom place la colonne telephone en troisième. Si vous aviez un aimant THIRD, il aurait eu le même effet.

id_contact	nom	telephone	prenom	email

Altération de table

La commande ALTER vous permet de presque tout modifier dans votre table sans avoir à réinsérer vos données. Mais faites attention, si vous modifiez le type de données d'une colonne, vous risquez de perdre vos données.

Altérations de Dataville

NOS SERVICES POUR LES TABLES EXISTANTES:

CHANGE change à la fois le nom et le type de données d'une colonne existante *

MODIFY change le type de données ou la position d'une colonne existante *

ADD ajoute une colonne à votre table, vous choisissez le type de données

DROP supprime une colonne de votre table *

Ce n'est qu'une petite altération, vous ne sentirez rien.

*** Risque de perte de données, pas de garantie.**

SERVICES SUPPLÉMENTAIRES

Réordonner vos colonnes

(disponible uniquement en utilisant ADD)

MUSCLEZ VOS NEURONES

Pourquoi cette table peut-elle avoir besoin d'être altérée ?

projekts

numero	descriptionduproj	entrepriseresp
1	peinture extérieure de la maison	Souchot
2	réaménagement de la cuisine	Forestier
3	pose du parquet	Keller
4	toiture	Voulzy

Rénovation extrême de tables

Commençons nos altérations avec une table ayant bien besoin d'une bonne rénovation.

Bienvenue à Rénovation extrême de Tables ! Dans les quelques pages à venir, nous allons transformer une table en mauvais état en quelque chose que n'importe quelle base de données serait fière d'accueillir.

Ceci ne nous en dit pas assez sur ce que cette table est censée contenir.

Ces noms de colonnes ne nous disent rien sur ce qu'elles contiennent.

Nous pouvons peut-être ajouter quelques tirets bas ici pour que ce soit plus lisible.

projekts

numero	descriptionduproj	entrepriseresp
1	peinture extérieure de la maison	Souchot
2	réaménagement de la cuisine	Forestier
3	pose du parquet	Keller
4	toiture	Voulzy

Même si les noms de la table et des colonnes ne sont pas géniaux, les données de la table sont valides et nous aimerions les garder.

Utilisons `DESCRIBE` pour voir comment cette table est construite. Cela nous montre si une colonne est la clé primaire et le type des données conservées dans chaque colonne.

Fichier Edition Fenêtre Aide MauvaiseConceptionDeTable

```
--> DESCRIBE projekts;
+-------------------+--------------+------+-----+---------+-------+
| Field             | Type         | Null | Key | Default | Extra |
+-------------------+--------------+------+-----+---------+-------+
| numero            | int(11)      | YES  |     | NULL    |       |
| descriptionduproj | varchar(100) | YES  |     | NULL    |       |
| entrepriseresp    | varchar(10)  | YES  |     | NULL    |       |
+-------------------+--------------+------+-----+---------+-------+
3 rows in set (0.01 sec)
```

Renommer la table

La table a quelques problèmes dans son état actuel mais, grâce à ALTER, nous allons la rendre apte à contenir une liste de projets d'améliorations requises pour une maison particulièrement en mauvais état. Notre première étape sera d'utiliser ALTER TABLE pour donner à notre table un nom significatif.

« projekts » est l'ancien nom de notre table.

```
ALTER TABLE projekts
RENAME TO liste_projets;
```

C'est assez clair en anglais : nous voulons RENOMMER notre table.

« liste_projets » est le nouveau nom que nous donnons à notre table.

Cette description va vous aider à trouver les autres façons d'utiliser ALTER dans la table. Trouvez, dans cette phrase, les colonnes décrivant notre future utilisation de la table, puis complétez les noms de colonnes.

id_projet

Pour avoir une table NORMALE, nous aurons aussi besoin d'une clé primaire contenant un numéro de projet unique. Puis nous aurons besoin de colonnes pour décrire chaque amélioration, sa date de début, l'estimation du coût et le nom de l'entreprise y travaillant, ainsi que leur numéro de téléphone.

Solutions des exercices

Cette description va vous aider à trouver les autres façons d'utiliser `ALTER` dans la table. Trouvez, dans cette phrase, les colonnes décrivant notre future utilisation de la table, puis complétez les noms de colonnes.

Ne vous inquiétez pas si vos noms de colonnes ne correspondent pas exactement à ceux-ci. Une certaine dose d'abréviations est acceptable tant que ce que vous stockez reste clair.

Assurez-vous que les noms courts, comme id_projet, sont compréhensibles pour vous et les autres personnes qui travaillent avec la base de données.

id_projet

date_debut

Pour avoir une table `NORMALE`, nous aurons aussi besoin d'une clé primaire contenant un numéro de projet unique. Puis nous aurons besoin de colonnes pour décrire chaque amélioration, sa date de début, l'estimation du coût et le nom de l'entreprise y travaillant, ainsi que leur numéro de téléphone.

desc_projet

tel_ent

estim_cout

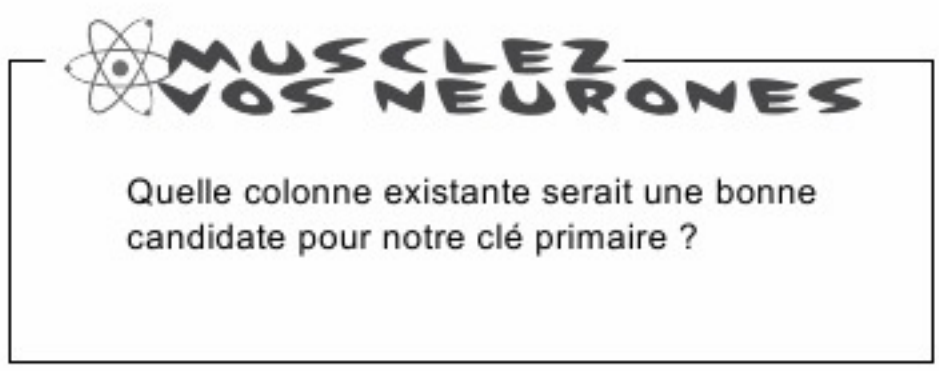
nom_ent

Nous devons faire des plans

liste_projets

numero	descriptionduproj	entrepriseresp
1	peinture extérieure de la maison	Souchot
2	réaménagement de la cuisine	Forestier
3	installation du parquet	Keller
4	toiture	Voulzy

Il semble que les données de trois de nos nouvelles colonnes soient déjà en place. Plutôt que de créer toutes les nouvelles colonnes, nous pouvons utiliser RENAME pour renommer nos colonnes existantes. En renommant ces colonnes qui contiennent des données valides, nous n'aurons pas besoin d'insérer les données dans de nouvelles colonnes.

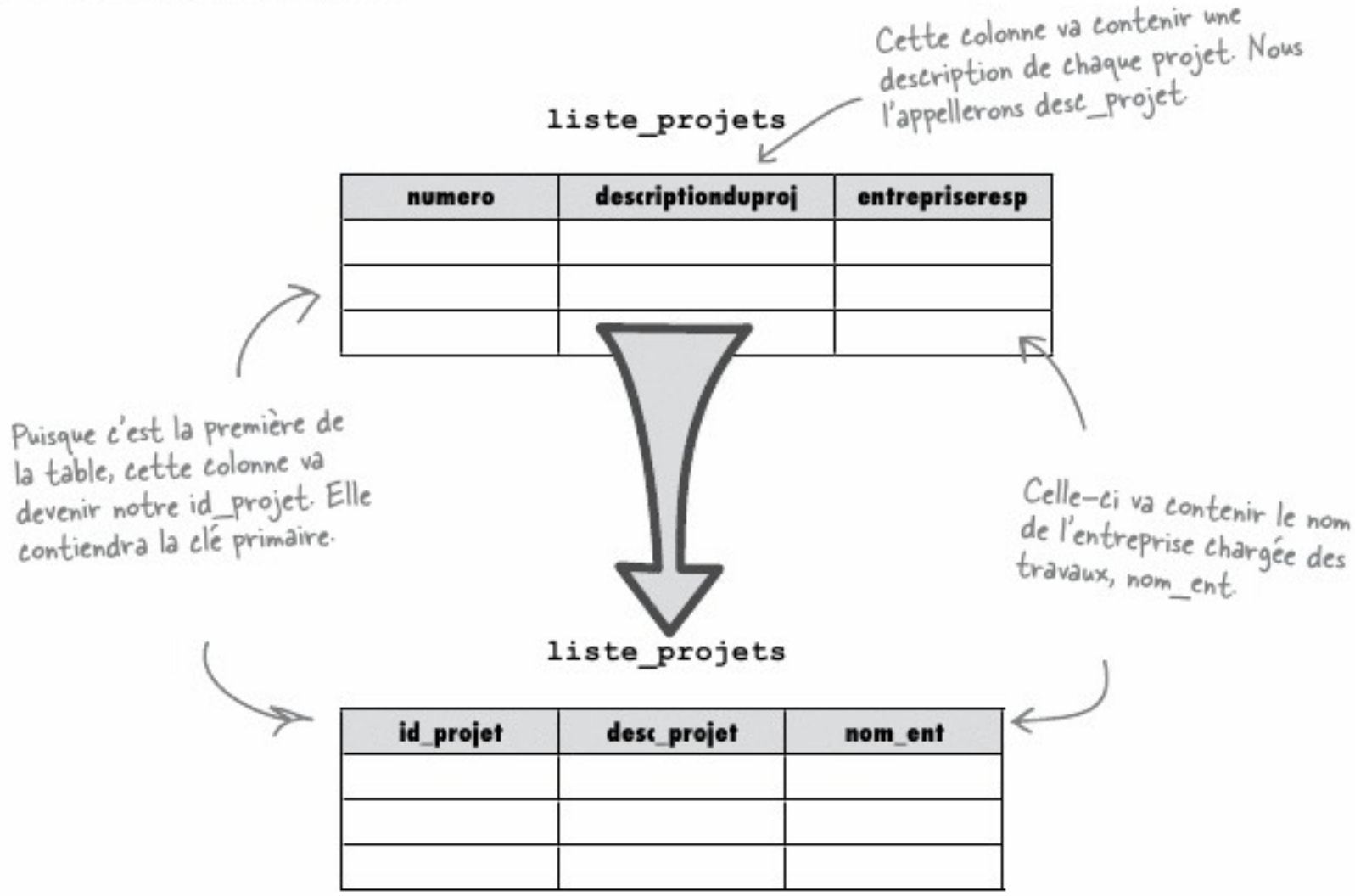


Retoucher nos colonnes

Nous avons maintenant un plan pour commencer et nous pouvons utiliser `ALTER` sur les colonnes déjà présentes dans notre table pour qu'elles correspondent avec trois de nos nouveaux noms de colonnes:

- `numero` est notre clé primaire : **`id_projet`**
- `descriptionduproj` est la description de chaque projet d'amélioration : **`desc_projet`**
- `entrepriseresp` est le nom de l'entreprise qui fait les travaux, **`nom_ent`** en plus court.

Cela nous laisse avec les trois colonnes appelées `estim_cout`, `tel_ent` et `date_debut` à ajouter.



Changement de structure

Nous avons décidé d'utiliser des colonnes existantes pour trois des colonnes dont nous avons besoin. Au-delà du simple changement de nom, nous devons regarder de plus près les types de données que conserve chacune de ces colonnes.

Voici la description que nous avons vue plus tôt.

```
Fichier  Edition  Fenêtre  Aide  MauvaiseConceptionDeTable
--> DESCRIBE projekts;
+-------------------+--------------+------+-----+---------+-------+
| Field             | Type         | Null | Key | Default | Extra |
+-------------------+--------------+------+-----+---------+-------+
| numero            | int(11)      | YES  |     | NULL    |       |
| descriptionduproj | varchar(100) | YES  |     | NULL    |       |
| entrepriseresp    | varchar(10)  | YES  |     | NULL    |       |
+-------------------+--------------+------+-----+---------+-------+
3 rows in set (0.01 sec)
```

GYM DU CERVEAU

Regardez le « Type » de chaque colonne et décidez si les types actuels sont adaptés aux futures données que nous pourrions conserver dans cette table.

ALTER et CHANGE

Pour notre prochaine étape, nous allons changer la colonne `numero` pour lui donner un nouveau nom, `id_projet` et lui ajouter `AUTO_INCREMENT`. Puis nous en ferons une clé primaire. Cela peut paraître compliqué mais ce ne l'est pas vraiment. En fait, vous pouvez faire tout cela en une seule commande :

Cette fois nous utilisons CHANGE COLUMN parce ~~que nous changeons~~ à la fois le nom et le type de données de la colonne anciennement appelée « numero ».

Nous utilisons toujours la même table mais rappelez-vous, nous lui avons donné un nouveau nom.

« id_projet » est le nouveau nom que nous voulons donner à notre colonne...

... et nous voulons la remplir avec des entiers auto incrémentés mais pas de valeurs NULL.

```
ALTER TABLE liste_projets
CHANGE COLUMN numero id_projet INT NOT NULL AUTO_INCREMENT,
ADD PRIMARY KEY (id_projet);
```

Voici la partie qui signale à notre logiciel SQL d'utiliser la colonne nouvellement appelée id_projet comme clé primaire.

À vos crayons !

Dessinez la table après avoir exécuté la commande ci-dessus.

Réponses page 233.

Changer deux colonnes en une instruction SQL

Nous allons modifier non pas une, mais deux colonnes en *une seule instruction*. Nous allons renommer les colonnes appelées `descriptionduproj` et `entrepriseresp`, et en même temps, nous allons changer leur type de données. Tout ce que nous avons à faire est d'inclure les deux lignes `CHANGE COLUMN` dans une instruction `ALTER TABLE` et de mettre une virgule entre les deux.

« descriptionduproj » est le nom de l'ancienne colonne que nous changeons dans cette commande.

« desc_projet » est le nouveau nom de la colonne.

Nous augmentons le nombre de caractères pour pouvoir avoir de plus longues descriptions.

```
ALTER TABLE liste_projets
CHANGE COLUMN descriptionduproj desc_projet VARCHAR(120),
CHANGE COLUMN entrepriseresp nom_ent VARCHAR(30);
```

L'ancien nom de l'autre colonne, « entrepriseresp », va également être changé...

... en nom_ent, et voici son nouveau type de données.

Ouvrez les yeux !

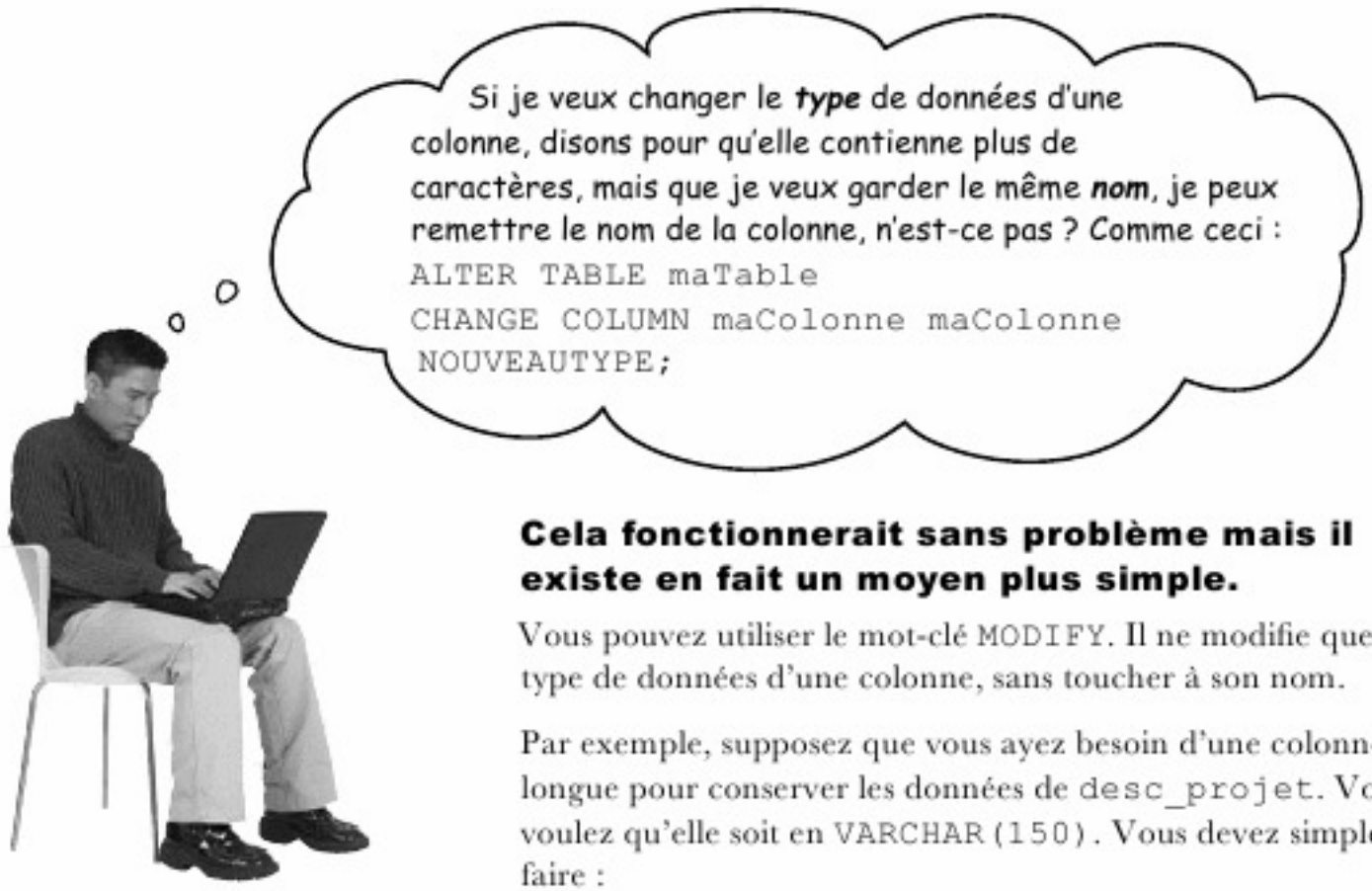
Si vous changez le type de données en quelque chose de nouveau, vous risquez de perdre des données.

Si le type de données que vous changez n'est pas compatible avec l'ancien type de données, votre commande ne sera pas exécutée et votre logiciel SQL vous dira qu'il y a une erreur dans votre instruction.

*Mais il y a pire : si les types **sont** compatibles, vos données peuvent être **tronquées**.*

Par exemple : si vous passez de varchar(10) à char(1), vos données vont changer de « Bozo » en simplement « B ».

C'est également valable pour les types numériques. Vous pouvez passer d'un type à un autre mais vos données seront converties au nouveau type et vous risquez de perdre une partie de vos données !



Cela fonctionnerait sans problème mais il existe en fait un moyen plus simple.

Vous pouvez utiliser le mot-clé `MODIFY`. Il ne modifie que le type de données d'une colonne, sans toucher à son nom.

Par exemple, supposez que vous ayez besoin d'une colonne plus longue pour conserver les données de `desc_projet`. Vous voulez qu'elle soit en `VARCHAR(150)`. Vous devez simplement faire :

```
ALTER TABLE liste_projets
MODIFY COLUMN desc_projet VARCHAR(150);
```

Le nom de la colonne que nous modifions.

Le nouveau type de données.

Bien sûr vous vous êtes assuré que le nouveau type de données ne va pas tronquer vos données !

il n'y a pas de

Questions stupides

Q: Mais si je veux changer l'ordre de mes colonnes ? Puis-je simplement écrire : ALTER TABLE liste_projets MODIFY COLUMN desc_projet AFTER nom_ent; ?

R: En fait, vous ne pouvez pas changer l'ordre des colonnes une fois que la table a déjà été créée. Le mieux que vous puissiez faire est d'ajouter une nouvelle colonne à l'endroit voulu et de faire un drop avec l'ancienne, mais vous perdrez toutes les données de l'ancienne colonne.

Q: Mais cela ne posera-t-il pas de problème si les colonnes ne sont pas conservées dans le bon ordre ?

R: Non, parce qu'heureusement, dans vos requêtes SELECT, vous pouvez préciser l'ordre dans lequel vos colonnes seront affichées dans les résultats de la requête. Peu importe l'ordre dans lequel vos données sont conservées sur votre disque dur, puisque vous pouvez exécuter ceci :

```
SELECT colonne3, colonne1 FROM votre_
table;
```

ou :

```
SELECT colonne1, colonne3 FROM votre_
table;
```

ou n'importe quel autre ordre, comme vous le voulez.

Exercices

liste_projets

id_projet	desc_projet	nom_ent
1		
2		
3		

Nous avons toujours besoin d'ajouter trois colonnes de plus : un **numéro de téléphone**, une **date de début** et une **estimation du coût**.

Écrivez ci-dessous une instruction ALTER TABLE unique pour le faire, en vous assurant de faire attention aux types de données. Puis complétez la table.

. .....

. .....

. .....

. .....

liste_projets

Solutions des exercices

Dites, je suis au téléphone avec mon agent. Continuez sans moi pour ajouter les colonnes restantes, d'accord ?

liste_projets

id_projet	desc_projet	nom_ent
1		
2		
3		

Nous avons toujours besoin d'ajouter trois colonnes de plus : un **numéro de téléphone**, une **date de début** et une **estimation du coût**.

Écrivez ci-dessous une instruction ALTER TABLE unique pour le faire, en vous assurant de faire attention aux types de données. Puis complétez la table.

Une VARCHAR de 10 caractères nous permet d'ajouter le code régional.

```
ALTER TABLE liste_projets
ADD COLUMN tel_ent VARCHAR(10),
ADD COLUMN date_debut DATE,
ADD COLUMN estim_cout DECIMAL(7,2);
```

Nous ajoutons de nouvelles colonnes, nous utilisons donc ADD.

Vous rappelez-vous de nos champs DEC ? Nous avons défini celui-ci pour qu'il ait 7 chiffres de long dont deux décimales.

liste_projets

id_projet	desc_projet	nom_ent	tel_ent	date_debut	estim_cout
1					
2					
3					

Vite ! Un DROP pour cette colonne

Arrêtez tout !

Nous venons de découvrir que notre projet a été suspendu. En conséquence, nous pouvons envoyer un drop dans notre colonne `date_debut`. Il n'y a pas d'intérêt à garder une colonne inutile qui prend de la place dans la base de données.

C'est une bonne habitude de programmation de ne garder que les colonnes dont vous avez besoin dans votre table. S'il y a une colonne que vous n'utilisez pas, envoyez-lui un drop. Avec `ALTER`, vous pouvez facilement la réajouter si vous en avez besoin à l'avenir.

Plus vous avez de colonnes, plus votre SGBDR doit travailler dur et plus votre base de données occupe de l'espace. Même si vous ne le remarquez pas avec une petite table, quand vos tables grandissent, les résultats ralentissent et le processeur de votre ordinateur doit travailler d'autant plus.

À vos crayons !

En fait, continuez et écrivez l'instruction SQL pour envoyer un drop à la colonne date_debut. Nous ne vous avons pas encore montré la syntaxe pour cela mais essayez quand même.

. .....

. .....

. .....

À vos crayons ! Solution

En fait, continuez et écrivez l'instruction SQL pour envoyer un drop à la colonne date_debut. Nous ne vous avons pas encore montré la syntaxe pour cela mais essayez quand même.

Voici notre liste_projets.

```
ALTER TABLE liste_projets
DROP COLUMN date_debut;
```

Si vous voulez jeter la colonne date_debut, vous pouvez utiliser la commande DROP. C'était facile !

La colonne à enlever de la table

Ouvrez les yeux !

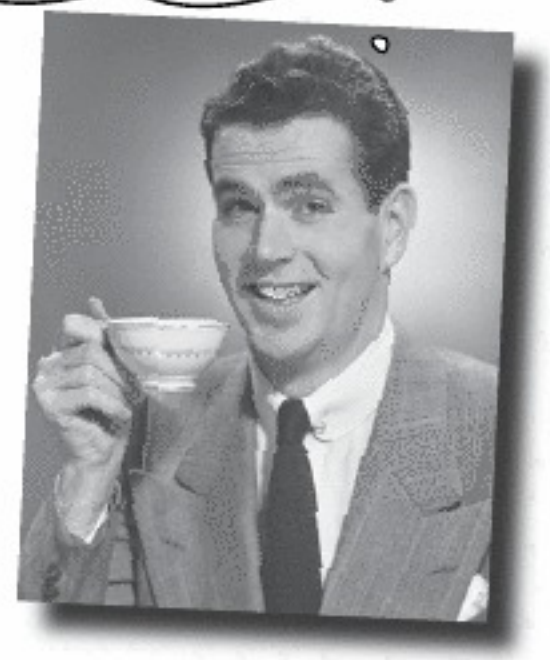
Une fois que vous avez jeté une colonne, tout ce qui s'y trouvait est aussi supprimé !

Utilisez DROP COLUMN avec prudence. Vous pouvez d'abord faire un SELECT de la colonne que vous voulez jeter pour être absolument sûr que vous voulez la jeter ! Il vaut mieux avoir des données en trop dans votre table qu'une donnée vitale qui manque.

Exercices

Tuner ma Table

Il est temps de transformer votre vieux tas de ferraille fatigué en un aimant à rendez-vous et de l'amener à un niveau de tuning de table dépassant tout ce que vous connaissez.



C'est simple. Grâce à `ALTER`, transformez cette pitoyable petite table « avant » contenant des données sur des voitures d'occasion en cette magnifique, rutilante table « après ». L'une des difficultés est de ne pas toucher aux données de la table, vous devez travailler autour. Êtes-vous à la hauteur ?

Des points de bonus si vous pouvez faire tout cela en une seule instruction `ALTER TABLE`.

Avant

vieilles_ferrailles

couleur	annee	marque	mo	combien
argent	1998	Porsche	Boxter	17992.540
NULL	2000	Jaguar	XJ	15995
rouge	2002	Cadillac	Escalade	40215.9

Après

table_voitures

id_voiture	VIN	marque	modele	couleur	annee	prix
1	RNKLK66N33G213481	Porsche	Boxter	argent	1998	17992.54
2	SAEDA44B175B04113	Jaguar	XJ	NULL	2000	15995.00
3	3GYEK63NT2G280668	Cadillac	Escalade	rouge	2002	40215.90

Solutions des exercices

Tuner ma Table

Il est temps de transformer votre vieux tas de ferraille fatigué en un aimant à rendez-vous et de l'amener à un niveau de tuning de table dépassant tout ce que vous connaissez.

C'est simple. Grâce à `ALTER`, transformez cette pitoyable petite table « avant » contenant des données sur des voitures d'occasion en cette magnifique, rutilante table « après ». L'une des difficultés est de ne pas toucher aux données de la table, vous devez travailler autour. Êtes-vous à la hauteur ?

Des points de bonus si vous pouvez faire tout cela en une seule instruction `ALTER TABLE`.

Avant

`vieilles_ferrailles`

couleur	annee	marque	mo	combien
argent	1998	Porsche	Boxter	17992.540
NULL	2000	Jaguar	XJ	15995
rouge	2002	Cadillac	Escalade	40215.9

Après

`table_voitures`

id_voiture	VIN	marque	modele	couleur	annee	prix
1	RNKLK66N33G213481	Porsche	Boxter	argent	1998	17992.54
2	SAEDA44B175B04113	Jaguar	XJ	NULL	2000	15995.00
3	3GYEK63NT2G280668	Cadillac	Escalade	rouge	2002	40215.90

Vous pouviez lancer un DESCRIBE avant pour voir les types de données et vous assurer de ne tronquer aucune donnée.

```
ALTER TABLE vieilles_ferrailles
RENAME TO table_voitures,
ADD COLUMN id_voiture INT NOT NULL AUTO_INCREMENT
FIRST,
ADD PRIMARY KEY (id_voiture),
ADD COLUMN VIN VARCHAR(17) SECOND,
CHANGE COLUMN mo modele VARCHAR(20),
MODIFY COLUMN couleur AFTER modele,
MODIFY COLUMN annee SIXTH,
CHANGE COLUMN combien prix DECIMAL(7,2);
```

Vous devez renommer la colonne « mo » en « modele » avant de déplacer les colonnes couleur et annee après elle.

Vous devez donner un type de données à la colonne renommée en « modele ».

Vous auriez aussi pu mettre « annee AFTER couleur » ou « annee BEFORE prix ».

Ouvrez les yeux !

Si vous utilisez mySQL, vous ne pouvez utiliser que les mots-clés FIRST et AFTER !

De plus, si vous voulez changer la position d'une colonne avec MODIFY, vous devez préciser le type de données de cette colonne. Enfin sachez que le mot-clé COLUMN y est optionnel, mais il peut vous aider à clarifier votre requête. Voyez si vous pouvez convertir cette requête en requête mySQL...

il n'y a pas de
Questions stupides

Q: Vous avez dit précédemment que je ne pouvais pas réordonner mes colonnes avec MODIFY mais mon outil logiciel SQL me laisse le faire. Comment fait-il ?

R: Votre logiciel exécute en fait un tas de commandes en coulisses. Il copie les valeurs de la colonne que vous souhaitez déplacer, les sauvegarde dans une table temporaire, envoie un drop à la colonne que vous voulez déplacer, altère votre table pour créer une nouvelle colonne avec le même nom que l'ancienne à la place où vous souhaitez qu'elle se trouve, rapatrie toutes les valeurs de la table temporaire dans votre nouvelle colonne et supprime la table temporaire.

Généralement, il vaut mieux ne pas toucher à la position de vos colonnes si elles contiennent déjà des données et que vous n'utilisez pas un logiciel pour réaliser toutes ces étapes pour vous. Vous pouvez SELECTionner vos colonnes dans l'ordre que vous voulez.

Q: Le seul moment où il est facile de changer l'ordre des colonne, c'est quand j'ajoute une nouvelle colonne ?

R: Exact. Il vaut mieux penser à l'ordre quand vous concevez la table, tout au début.

Q: J'ai accidentellement créé une clé primaire mais j'ai changé d'avis et je veux utiliser une autre colonne, que faire ? Y a-t-il un moyen d'enlever la désignation de la clé primaire sans changer les données qui s'y trouvent ?

R: Il y en a un et il est simple, mais il ne fonctionne que si votre clé primaire n'est pas (ou plus) auto incrémentée :

```
ALTER TABLE votre_table DROP PRIMARY
KEY;
```

Q: Et AUTO_INCREMENT ?

R: Vous pouvez l'ajouter à une colonne qui ne l'a pas de cette façon :

```
ALTER TABLE votre_table CHANGE votre_
id votre_id INT(11) NOT NULL AUTO_
INCREMENT;
```

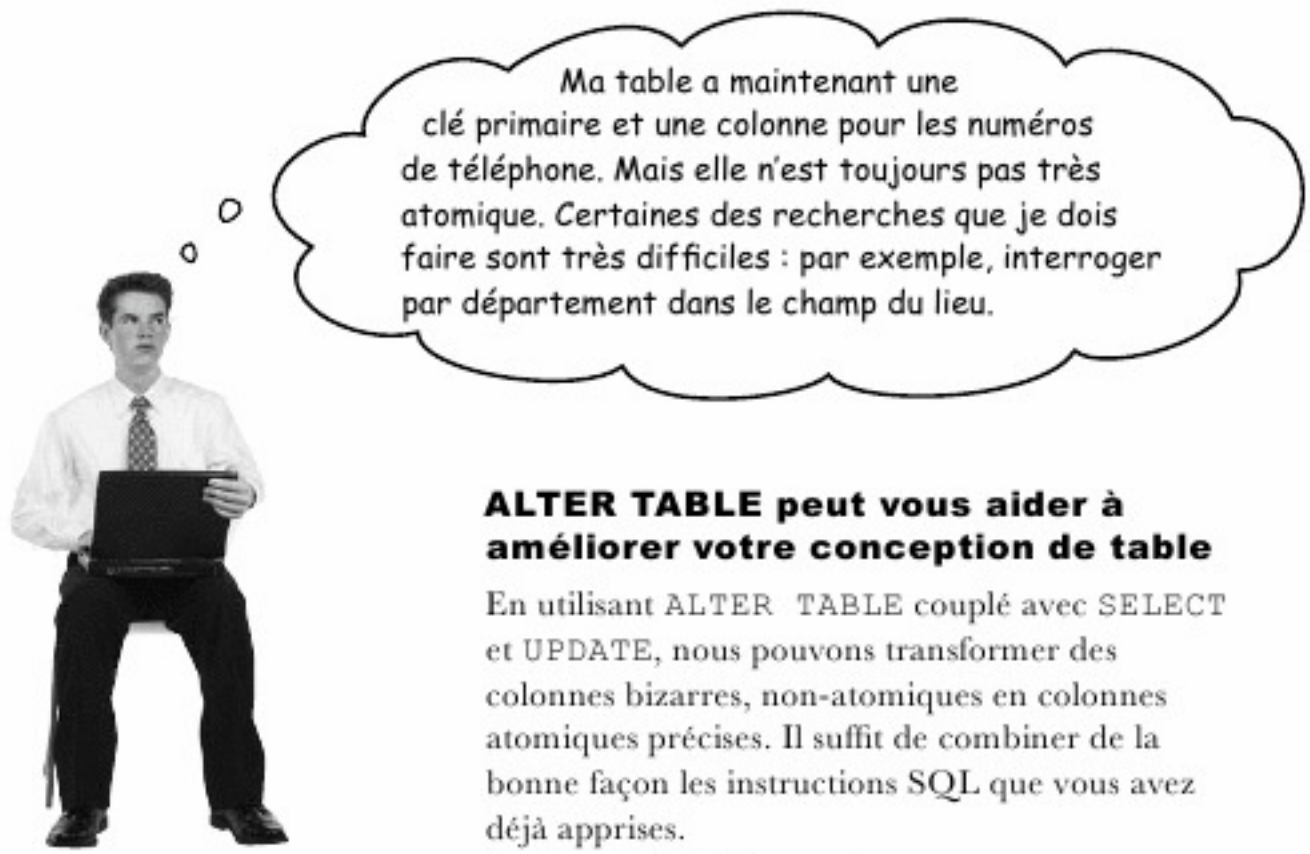
Et vous pouvez l'enlever comme ceci :

```
ALTER TABLE votre_table CHANGE votre_id
votre_id INT(11) NOT NULL;
```

Il est important de garder à l'esprit que vous ne pouvez avoir qu'un champ AUTO_INCREMENT par table, que ce doit être un type de données entier (INTEGER), qu'il ne peut pas contenir de NULL et que ce doit être une clé primaire.

POINTS D'IMPACT

- Utilisez `CHANGE` si vous voulez changer à la fois le nom et le type de données d'une colonne.
- Utilisez `MODIFY` si vous ne voulez changer que le type de données.
- `DROP COLUMN` jette une colonne : il supprime de la table la colonne nommée.
- Utilisez `RENAME` pour changer le nom de votre table.
- Vous pouvez changer l'ordre de vos colonnes en utilisant `FIRST`, `LAST`, `BEFORE nom_colonne`, `AFTER nom_colonne`, `SECOND`, `THIRD`, `FOURTH`, etc.
- Avec certains SGBDR, vous ne pouvez changer l'ordre des colonnes d'une table que lorsque vous les ajoutez à la table.



ALTER TABLE peut vous aider à améliorer votre conception de table

En utilisant `ALTER TABLE` couplé avec `SELECT` et `UPDATE`, nous pouvons transformer des colonnes bizarres, non-atomiques en colonnes atomiques précises. Il suffit de combiner de la bonne façon les instructions SQL que vous avez déjà apprises.

Jetons un œil à l'instruction `CREATE TABLE` pour la table `mes_contacts` de Greg.

```
CREATE TABLE mes_contacts
(
    id_contact INT NOT NULL AUTO_INCREMENT
    nom VARCHAR(30) default NULL,
    prenom VARCHAR(20) default NULL,
    email VARCHAR(50) default NULL,
    sexe CHAR(1) default NULL,
    anniversaire DATE default NULL,
    profession VARCHAR(50) default NULL,
    lieu VARCHAR(50) default NULL,
    statut VARCHAR(20) default NULL,
    centres_interet VARCHAR(100) default NULL,
    cherche VARCHAR(100) default NULL,
    PRIMARY KEY (id_contact)
);
```

Nous avons ajouté ces deux lignes pour créer et désigner notre clé primaire.

Ces quatre colonnes ne sont pas très atomiques et pourraient profiter de quelques corrections avec ALTER TABLE.

La colonne lieu non-atomique vue de plus près

Parfois, Greg veut seulement connaître le département ou la ville de quelqu'un, la colonne `lieu` est donc une bonne candidate à diviser en deux colonnes. Voyons à quoi ressemblent les données de la colonne :

```
Fichier Edition Fenêtre Aide LieuLieuLieu
--> SELECT lieu FROM mes_contacts;
+-----------------+
|  lieu           |
+-----------------+
| Nice, 06        |
| Rennes, 35      |
| St-Malo, 35     |
| Nantes, 44      |
| La Rochelle, 17 |
| Orléans, 45     |
```

Une partie des données de la colonne lieu de la table mes_contacts.

Nice, 06
Rennes, 35
St-Malo, 35
Nantes, 44
La Rochelle, 17

Deux chiffres pour le département.

Nom de la ville.

Une virgule.

Ces données sont formatées de façon systématique. D'abord le nom de la ville suivi par une virgule puis un département en deux chiffres. Comme les données sont fidèles à cette syntaxe, nous pouvons séparer la ville du département.

GYM DU CERVEAU

Pourquoi voulons-nous séparer la ville du département ?

Qu'allons-nous faire ensuite, à votre avis ?

Recherche de motifs

Toute la colonne lieu de la table `mes_contacts` suit le même modèle : Nom de ville suivi par une virgule, puis les deux chiffres du département. Le fait que ce soit systématique et suive un motif va nous aider à le diviser pour que ce soit plus atomique.

Nom de ville, XX

Ce deux derniers caractères contiennent le numéro du département. Si nous ajoutons une colonne dpmt à notre table, ce sont ces données que nous voudrions y mettre.

Cette virgule qui se trouve toujours devant le numéro du département pourrait s'avérer utile...

Nous pouvons prendre tout ce qui se trouve avant la virgule pour le mettre dans une colonne contenant les `noms de villes`.

Et nous pouvons prendre les deux derniers caractères de notre colonne lieu pour les mettre dans une nouvelle colonne appelée `dpmt`.

Nom de ville , XX

Nous avons besoin d'une fonction qui nous permette de prendre tout ce qui se trouve avant la virgule...

... et nous avons besoin d'une fonction qui va prendre les deux derniers caractères.

À vos crayons !

Écrivez une instruction ALTER TABLE pour ajouter les colonnes ville et dpmt à mes_contacts.

. .....

. .....

. .....

ALTER TABLE mes_contacts
ADD COLUMN ville VARCHAR(50),
ADD COLUMN dpmt INT(2);

Quelques fonctions de chaînes pratiques

Nous avons identifié deux motifs. Nous avons maintenant besoin de récupérer les numéros de département et de les ajouter dans une nouvelle colonne **dpmt**. Il nous faut également récupérer ce qui se trouve avant la virgule pour une colonne **ville**. Après avoir créé nos nouvelles colonnes, voici comment nous pouvons extraire les valeurs dont nous avons besoin :

Les valeurs texte et les valeurs conservées dans des colonnes CHAR ou VARCHAR sont appelées des chaînes.

Pour SELECTionner les deux derniers caractères

Utilisez `RIGHT()` et `LEFT()` pour sélectionner **un nombre précis de caractères** d'une colonne.

```
SELECT RIGHT(lieu, 2) FROM mes_contacts;
```

Commence du côté droit (RIGHT) de la colonne. (Vous pouvez utiliser LEFT exactement de la même façon pour commencer du côté gauche).

Voici la colonne à utiliser.

Voici le nombre de caractères à sélectionner en partant du côté droit de la colonne.

Les fonctions de chaînes vous permettent de sélectionner une partie d'une colonne de texte.

Pour SELECTionner tout ce qu'il y a avant la virgule

Utilisez `SUBSTRING_INDEX()` pour récupérer une partie de la colonne, soit une sous-chaîne (*substring*). Cela récupèrera tout ce qui se trouve **devant** un **caractère ou une chaîne donnée**. Nous pouvons donc mettre notre virgule entre guillemets simples et `SUBSTRING_INDEX()` sélectionnera tout ce qui se trouve devant elle.

```
SELECT SUBSTRING_INDEX(lieu, ',', 1) FROM mes_contacts;
```

Ceci prend une partie de la colonne, soit une sous-chaîne (substring). Cela cherche la chaîne entre les guillemets simples (dans ce cas, une virgule) et prend tout ce qui se trouve devant.

De nouveau, le nom de la colonne.

Voici la virgule que cherche la commande.

Voici la partie difficile. C'est « 1 » parce que l'on cherche la première virgule. Si c'était « 2 », on continuerait jusqu'à la deuxième virgule pour prendre tout ce qui se trouve devant elle.

À essayer à la maison

SQL possède beaucoup de fonctions qui vous permettent de manipuler des valeurs de chaînes dans vos tables. Les chaînes sont conservées dans des colonnes de texte, le plus souvent des types de données `VARCHAR` ou `CHAR`.

Voici une liste de quelques-unes des fonctions de chaînes les plus courantes et les plus utiles. Essayez-les vous-même en tapant les instructions `SELECT`.

`SUBSTRING(votre_chaîne, point_de_départ, longueur)` vous donne une partie de `votre_chaîne`, commençant à la lettre du `point_de_départ`. `longueur` définit la longueur de la chaîne que vous récupérez.

```
SELECT SUBSTRING('Saint-Malo, 35', 7, 4);
```

`UPPER(votre_chaîne)` et **`LOWER(votre_chaîne)`** mettra toute votre chaîne respectivement en majuscules ou en minuscules.

```
SELECT UPPER('uSa');

SELECT LOWER('spaGHEtti');
```

`REVERSE(votre_chaîne)` inverse l'ordre des lettres de votre chaîne.

```
SELECT REVERSE('spaGHEtti');
```

`LTRIM(votre_chaîne)` et **`RTRIM(votre_chaîne)`** renvoie votre chaîne en enlevant les espaces avant (à gauche, d'où le L de *left*) ou après (à droite, d'où le R de *right*) une chaîne.

```
SELECT LTRIM(' nourriture du chien ');

SELECT RTRIM(' nourriture du chat ');
```

`LENGTH(votre_chaîne)` renvoie le nombre de caractères de votre chaîne.

```
SELECT LENGTH('Saint-Malo, 35 ');
```

IMPORTANT : les fonctions de chaînes ne changent PAS les données de votre table ; elles renvoient seulement les chaînes altérées comme résultat de requête.

À QUOI JE SERS ?

Nous essayons de récupérer les informations de notre colonne lieu pour les tranférer dans deux nouvelles colonnes, ville et dpmt.

Voici les étapes que nous allons suivre pour cela. Reliez chaque étape au(x) mot(s)-clé(s) SQL dont nous avons besoin pour la réaliser.

SUBSTRING_INDEX()

SELECT

1. Étudier les données d'une colonne donnée pour trouver un motif.

LEFT

ADD COLUMN

2. Ajouter de nouvelles colonnes vides dans notre table.

ADJUST

RIGHT

3. Récupérer une partie des données d'une colonne de texte.

ALTER TABLE

DELETE

4. Placer les données récupérées à l'étape 3 dans l'une des colonnes vides.

UPDATE

INSERT

→ Réponses page 228.

Avec ce que nous savons jusqu'ici, nous devrions écrire une instruction UPDATE pour un enregistrement à la fois, avec un SELECT pour obtenir les bonnes données.

Mais avec SQL, nous pouvons combiner nos instructions. Tournez la page pour voir comment mettre les valeurs dans nos nouvelles colonnes.

À QUOI JE SERS ?

Nous essayons de récupérer les informations de notre colonne lieu pour les tranférer dans deux nouvelles colonnes, ville et dpmt.

Voici les étapes que nous allons suivre pour cela. Reliez chaque étape au mot(s)-clé(s) SQL dont nous avons besoin pour la réaliser.

SUBSTRING_INDEX()

SELECT

1. Étudier les données d'une colonne donnée pour trouver un motif.

ADD COLUMN

LEFT

2. Ajouter de nouvelles colonnes vides dans notre table.

ADJUST

RIGHT

3. Récupérer une partie des données d'une colonne de texte.

ALTER TABLE

DELETE

4. Placer les données récupérées à l'étape 3 dans l'une des colonnes vides.

INSERT

UPDATE

Utiliser une colonne pour en remplir une nouvelle

Vous souvenez-vous de notre syntaxe pour `UPDATE` ? Nous pouvons l'utiliser pour placer la même nouvelle valeur dans toutes les lignes de notre table. L'instruction ci-dessous montre la syntaxe pour **changer la valeur de toutes les lignes dans une colonne**. Au lieu de `nouvellevaleur`, vous pouvez mettre une valeur ou un autre nom de colonne.

```
UPDATE nom_table
SET nom_colonne = nouvellevaleur;
```

Chaque ligne de la table est remplie, une à la fois, avec cette valeur.

Pour ajouter des données à nos nouvelles colonnes `ville` et `dpmt`, nous pouvons utiliser la fonction de chaîne `RIGHT()` dans cette instruction `UPDATE`. La fonction de chaîne récupère les deux derniers caractères de l'ancienne colonne `lieu` et les place dans la nouvelle colonne `dpmt`.

```
UPDATE mes_contacts
SET dpmt = RIGHT(lieu, 2);
```

Voici la nouvelle colonne de nos données sur les départements.

Et voici la fonction de chaîne qui récupère les deux derniers caractères de la colonne lieu.

Cela fonctionne sans clause WHERE. Tournez la page pour voir comment.

Comment fonctionnent nos combinaisons d'UPDATE et de SET

Votre logiciel SQL interprète l'instruction pour chaque ligne de la table une par une puis il revient en arrière et recommence jusqu'à ce que tous les numéros de départements soient arrivés dans leur nouvelle colonne `dpmt`.

mes_contacts

id_contact	lieu	ville	dpmt
1	Nice, 06		
2	Rennes, 35		
3	Orléans, 45		

Voici une version simplifiée de notre table.

```
UPDATE mes_contacts
SET dpmt = RIGHT(lieu, 2);
```

Et voici notre instruction SQL.

Voyons comment cela fonctionne dans cette table exemple. Au premier passage, le logiciel prend le lieu pour la première ligne et opère dessus.

Puis il recommence à parcourir la table en entier une deuxième fois, trouve le lieu dans la deuxième ligne, opère dessus, etc., jusqu'à ce que tous les enregistrements de lieux soient répartis et qu'il n'y ait plus d'enregistrement qui corresponde à l'instruction.

Vous pouvez utiliser des fonctions de chaînes combinées avec SELECT, UPDATE et DELETE.

Premier passage

```
UPDATE mes_contacts
SET dpmt = RIGHT('Nice, 06',2)
```

Prend le premier enregistrement de la colonne lieu et opère dessus.

Deuxième passage

```
UPDATE mes_contacts
SET dpmt = RIGHT('Rennes, 35',2)
```

Maintenant le deuxième.

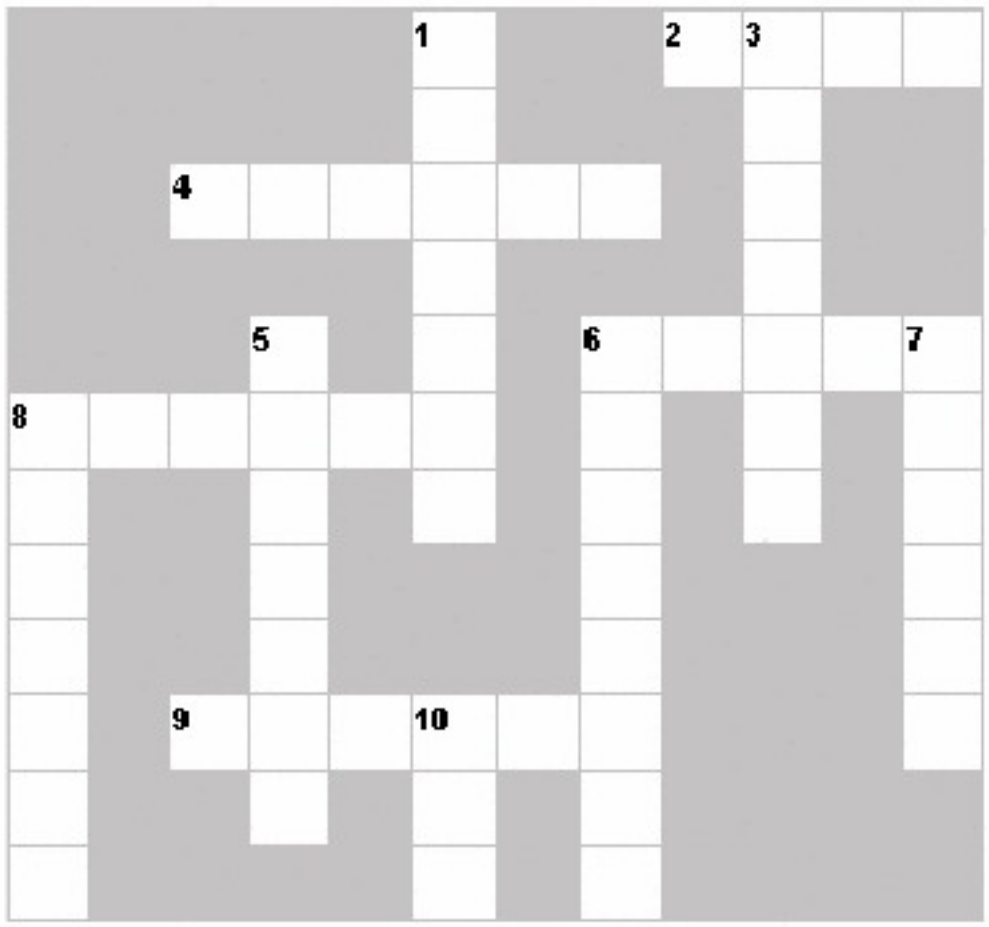
Troisième et dernier passage, puisqu'il n'y a que trois enregistrements

```
UPDATE mes_contacts
SET dpmt = RIGHT('Orléans, 45',2)
```

Et finalement le troisième.

Alter-croisés

Comment des mots-croisés vous aident-ils à apprendre SQL ? Hé bien, ils vous font réfléchir aux commandes et aux mots-clés de ce chapitre d'une manière différente.



Définitions horizontales :

2. Si vous n'avez plus besoin d'une colonne, utilisez ... COLUMN avec ALTER.

4. ALTER TABLE Projekts ... TO liste_projets.

6. ... (votre_chaîne) renvoie votre chaîne sans les espaces en trop situés avant (à gauche de) la chaîne.

8. Utilisez ce mot-clé pour altérer le type de données conservé dans une colonne.

9. Utilisez ... pour changer le nom de votre table.

Définitions verticales :

1. Vous pouvez utiliser des fonctions de ... combinées avec SELECT, UPDATE et DELETE.

3. ... (votre_chaîne) inverse l'ordre des lettres de votre chaîne.

5. Vous ne pouvez avoir qu'un champ AUTO_INCREMENT par table et ce doit être un type de données ... .

6. SUBSTRING (votre_chaîne, point_de_départ, longueur) vous donne une partie de votre_chaîne commençant par la lettre à la position point_de_départ. ... définit la longueur de la chaîne que vous récupérez.

7. Utilisez cette clause avec ALTER si vous ne voulez changer que le type de données.

8. Les valeurs conservées dans les colonnes CHAR ou VARCHAR s'appellent ainsi.

10. Notre table peut recevoir de nouvelles colonnes avec l'instruction ALTER et la clause ... COLUMN.

Boîte à outils SQL

Félicitez-vous. Vous maîtrisez le chapitre 5 et vous venez d'ajouter ALTER à votre boîte à outils. Pour une liste exhaustive des outils du livre, voyez l'annexe C.

ALTER TABLE

Vous permet de changer le nom de votre table et sa structure toute entière tout en gardant les données qui s'y trouvent.

ALTER avec ADD

Vous permet d'ajouter une colonne à votre table et de choisir où la placer.

ALTER avec DROP

Vous permet d'envoyer un drop à une colonne de votre table pour la supprimer.

ALTER avec CHANGE

Vous permet de changer à la fois le nom et le type de données d'une colonne existante.

ALTER avec MODIFY

Vous permet de ne changer que le type de données d'une colonne existante.

Fonctions de chaînes

Vous permettent de modifier des copies du contenu des colonnes de chaînes quand elles sont renvoyées par une requête. Les valeurs originales restent inchangées.

À vos crayons !

Solution
De la page 210.

Dessinez la table après avoir exécuté la commande de la page 210.

`liste_projets`

L'ancien « numero » est devenu id_projet et cette colonne contient les valeurs auto incrementées de la clé primaire.

id_projet	descriptionduproj	entrepriseresp
1	peinture extérieure de la maison	Souchot
2	réaménagement de la cuisine	Forestier
3	pose du parquet	Keller
4	toiture	Voulzy

Solution de l'Alter-croisés

					1 C			2 D	3 R	O	P
					H				E		
		4 R	E	N	A	M	E		V		
					Î				E		
			5 I		N		6 L	T	R	I	7 M
8 C	H	A	N	G	E		O		S		O
H			T		S		N		E		D
A			E				G				I
Î			G				U				F
N		9 R	E	N	10 A	M	E				Y
E			R		D		U				
S					D		R				

6 SELECT avancé

Un regard neuf sur vos données

Il est temps d'ajouter un peu de finesse dans votre boîte à outils. Vous savez déjà comment SELECTionner des données et utiliser des clauses WHERE. Mais parfois il faut plus de **précision** que ne peuvent en apporter SELECT et WHERE. Dans ce chapitre, vous apprendrez à **ordonner et regrouper** vos données, ainsi qu'à **lancer des opérations mathématiques** sur vos résultats.

Dataville Vidéo se réorganise

Le propriétaire de Dataville Vidéo a un magasin mal organisé. Dans son système actuel, les films peuvent atterrir sur différentes étagères selon l'employé qui les met en rayon. Il a commandé de nouvelles étagères et pense qu'il est grand temps de définir une fois pour toutes ses catégories de films.

À : Employés de Dataville Vidéo
De : Le Patron
Sujet : De nouvelles étagères impliquent de nouvelles catégories !

Salut à l'équipe,

Les nouvelles étagères sont arrivées, vous allez donc organiser nos films. Nous pouvons utiliser les noms de genres suivants :

Action et Aventure
Drame
Comédie
Familial
Horreur
SF et Fantastique
Divers

Je vous laisse le soin de trouver comment faire fonctionner notre table actuelle avec ces nouveaux genres.

On se voit pour déjeuner,

Votre patron

Dans le système actuel, on utilise des valeurs vrai et faux pour les genres de films. Il est donc difficile de trouver où catégoriser les films. Par exemple, si un film a V pour comédie mais aussi pour SF, où doit-il être rangé ?

« V » et « F » sont les abréviations de Vrai et Faux.

Voilà quand le magasin a acquis une copie.

table_films

id_film	titre	public	drame	comedie	action	gore	SF	pour_enfants	dessin_anime	achat
1	Monstres & Cie	TP	F	V	F	F	F	V	V	06-03-2002
2	Le parrain	+17	F	F	V	V	F	F	F	05-02-2001
3	Autant en emporte le vent	TP	V	F	F	F	F	F	F	20-11-1999
4	American Pie	+17	F	V	F	F	F	F	F	19-04-2003
5	Les griffes de la nuit	+17	F	F	V	V	F	F	F	19-04-2003
6	Casablanca	AP	V	F	F	F	F	F	F	05-02-2001

Toutes ces colonnes permettent de répondre aux questions des clients sur le contenu d'un film donné.

Les problèmes de notre table actuelle

Voici un condensé des problèmes que Dataville Vidéo rencontre avec la table actuelle.

Quand les films sont rendus, on ne sait pas où ils doivent aller.

Si nous avons des valeurs V pour plusieurs colonnes de la table, il n'y a pas de moyen evident de savoir où ce film doit être placé. Les films doivent toujours être associés à un **seule genre**.

Le sujet du film n'est pas clair pour les gens.

Nos clients sont déroutés quand ils voient une couverture sanglante dans la section comédie. Pour l'instant, aucune de nos valeurs V/F n'a priorité sur les autres quand les films sont rangés.

Ajouter des données Vrai et Faux est chronophage et des erreurs arrivent souvent.

À chaque fois qu'un nouveau film arrive, il doit être inséré avec toutes ces colonnes V/F. Plus on en saisit, plus il y a d'erreurs qui surgissent. Parfois une colonne qui doit être V est accidentellement entrée comme F et inversement. Une colonne genre nous aiderait à vérifier nos colonnes V/F et finalement nous pourrions nous débarrasser de ces V/F par la même occasion.

Ce qu'il nous faut ici, c'est une colonne genre pour accélérer la mise en rayon, aider les clients à comprendre quel type de film ils louent et limiter les erreurs dans nos données.

MUSCLEZ VOS NEURONES

Comment réorganiseriez-vous les colonnes actuelles en nouveaux **genres** ?
Y a-t-il des films qui pourraient correspondre à plus d'un des nouveaux genres ?

Faire correspondre les données existantes

Vous savez comment `ALTERer` votre table pour y ajouter la nouvelle colonne catégorie mais ajouter les catégories elles-mêmes est un peu plus ardu. Heureusement, les données qui sont déjà dans la table peuvent nous aider à trouver le genre de chaque film sans que nous ayons besoin de tous les regarder.

Réécrivons les relations avec des phrases simples :

Si cette colonne est 'V' : **drame** nous définissons la colonne genre comme 'drame'

Si cette colonne est 'V' : **comedie** nous définissons la colonne genre comme 'comédie'

Si cette colonne est 'V' : **action** nous définissons la colonne genre comme 'action'

Si cette colonne est 'V' : **gore** nous définissons la colonne genre comme 'horreur'

Si cette colonne est 'V' : **SF** nous définissons la colonne genre comme 'SF'

Si cette colonne est 'V' : **pour_enfants** nous définissons la colonne genre comme 'familial'

Si cette colonne est 'V' : **dessin_anime** et que cette colonne est 'TP' : **public** nous définissons la colonne genre comme 'familial'

Si cette colonne est 'V' : **dessin_anime** et que cette colonne n'est **PAS** 'TP' : **public** nous définissons la colonne genre comme 'divers'

Tous les dessins animés ne sont pas pour les enfants. La colonne public vous aide à déterminer si un film appartient ou pas au genre familial, si elle est vraie ou fausse. Si la classification est TP (tout public), nous pouvons l'appeler 'familial'; sinon, nous l'appelerons 'divers'.

Peupler la nouvelle colonne

Nous pouvons maintenant traduire ces phrases en instructions UPDATE :

```
UPDATE table_films SET genre = 'drame' where drame = 'V';
UPDATE table_films SET genre = 'comédie' where comedie = 'V';
UPDATE table_films SET genre = 'action' where action = 'V';
UPDATE table_films SET genre = 'horreur' where gore = 'V';
UPDATE table_films SET genre = 'SF' where SF = 'V';
UPDATE table_films SET genre = 'familial' where pour_enfants = 'V';
UPDATE table_films SET genre = 'familial' where dessin_anime = 'V' AND public = 'TP';
UPDATE table_films SET genre = 'divers' where dessin_anime = 'V' AND public <> 'TP';
```

public est différent de 'TP'.

À vos crayons !

Complétez la valeur genre pour ces films.

table_films

titre	public	drame	comedie	action	gore	SF	pour_ enfants	dessin_ anime	genre
Grande aventure	TP	F	F	F	F	F	V	F	
Greg : histoire secrète	AP	F	F	V	F	F	F	F	
Clowns fous, Les	+17	F	F	F	V	F	F	F	
Paraskavedekatriaphobie	+17	V	V	V	F	V	F	F	
Rat nommé Alice, Un	TP	F	F	F	F	F	V	F	
Fin de ligne	+17	V	F	F	V	V	F	V	
Choses qui brillent, Les	AP	V	F	F	F	F	F	F	
Reprends-le	+17	F	V	F	F	F	F	F	
Appât pour requin	TP	F	F	F	F	F	V	F	
Pirate en colère	AP	F	V	F	F	F	F	V	
Planète potentiellement habitable	AP	F	V	F	F	V	F	F	

L'ordre dans lequel nous évaluons chaque colonne V/F importe-t-il ? .....

À vos crayons ! Solution

Complétez la valeur genre pour ces films.

table_films

titre	public	drame	comedie	action	gore	SF	pour_ enfants	dessin_ anime	genre
Grande aventure	TP	F	F	F	F	F	V	F	familial
Greg : histoire secrète	AP	F	F	V	F	F	F	F	action
Clowns fous, Les	+17	F	F	F	V	F	F	F	horreur
Paraskavedekatriaphobie	+17	V	V	V	F	V	F	F	?
Rat nommé Alice, Un	TP	F	F	F	F	F	V	F	familial
Fin de ligne	+17	V	F	F	V	V	F	V	?
Choses qui brillent, Les	AP	V	F	F	F	F	F	F	drame
Reprends-le	+17	F	V	F	F	F	F	F	comédie
Appât pour requin	TP	F	F	F	F	F	V	F	familial
Pirate en colère	AP	F	V	F	F	F	F	V	divers
Planète potentiellement habitable	AP	F	V	F	F	V	F	F	?

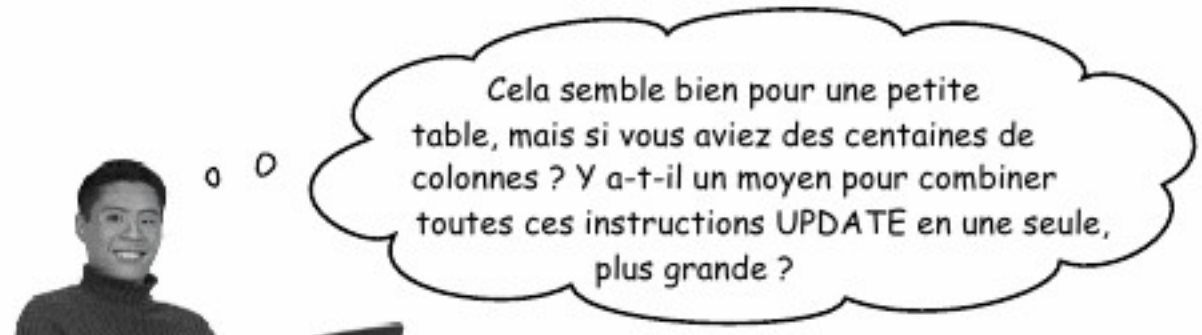
Le point d'interrogation signifie qu'une colonne a été changée par plus d'une UPDATE.
Cette valeur change selon l'ordre dans lequel les UPDATE sont exécutées.

L'ordre dans lequel nous évaluons chaque colonne V/F importe-t-il ? Oui, il est important.

L'ordre est important

Par exemple, si nous parcourons les colonnes dans l'ordre, « Paraskavedekatriaphobie » finirait classé comme SF, même s'il est possible que ce soit plutôt une comédie. Nous ne savons pas s'il doit être considéré comme comédie, action, drame ou SF. Comme sa classification n'est pas claire, il est sans doute mieux de le mettre dans la catégorie divers.

L'ordre est important.
Deux instructions UPDATE peuvent changer la valeur de la même colonne.



Hé bien, vous pouvez écrire une grande instruction UPDATE, mais il existe un meilleur moyen.

L'expression `CASE` combine toutes les instructions, mot-clé `UPDATE` en confrontant la valeur d'une colonne existante à une condition. Si elle remplit la condition, la nouvelle colonne reçoit une valeur spécifiée.

Cela vous permet même de dire à votre SGBDR ***que faire si des enregistrements ne remplissent pas les conditions***.

Voici sa syntaxe de base :

```
UPDATE ma_table
SET nvelle_colonne =
CASE
  WHEN colonne1 = valeur1
    THEN nvellevaleur1
  WHEN colonne2 = valeur2
    THEN nvellevaleur2
  ELSE nvellevaleur3
END;
```

La valeur de la colonne que vous indiquez ici va être changée en la valeur appropriée ci-dessous.

Ceci débute l'expression CASE.

WHEN (si) cette condition est remplie...

THEN (alors) définir la valeur de nvelle_colonne à cette valeur.

WHEN (si) une autre condition est remplie...

THEN (alors) définir la valeur de nvelle_colonne à cette autre valeur.

Les alinéas ne changent rien à l'expression ; ils la rendent juste plus facile à comprendre quand vous regardez le code.

Tout ce qui ne remplit aucune des conditions prend cette valeur à la place.

Ceci termine l'expression CASE et l'instruction UPDATE tout entière (parce que c'est suivi d'un point-virgule).

UPDATE avec une expression CASE

Voyons l'expression CASE à l'œuvre dans notre table_films.

```
UPDATE table_films
SET genre =
CASE
  WHEN drame = 'V' THEN 'drame'
  WHEN comedie = 'V' THEN 'comédie'
  WHEN action = 'V' THEN 'action'
  WHEN gore = 'V' THEN 'horreur'
  WHEN SF = 'V' THEN 'SF'
  WHEN pour_enfants = 'V' THEN 'familial'
  WHEN dessin_anime = 'V' THEN 'familial'
  ELSE 'divers'
END;
```

C'est la même chose que de dire UPDATE table_films SET genre = 'drame' WHERE drame = 'V', mais avec beaucoup moins à taper !

Les valeurs qui étaient inconnues quand nous utilisions UPDATE tout seul pour peupler la nouvelle colonne ont maintenant des valeurs pour genre.

Tout ce qui ne correspond pas aux conditions des lignes ci-dessus reçoit une valeur 'divers' pour genre.

Mais remarquez que nous avons également de nouvelles valeurs pour « Pirate en colère » et « Fin de ligne ».

table_films

titre	public	drame	comedie	action	gore	SF	pour_ enfants	dessin_ anime	genre
Grande aventure	AP	F	F	F	F	F	F	V	familial
Greg : histoire secrète	AP	F	F	V	F	F	F	F	action
Clowns fous, Les	+17	F	F	F	V	F	F	F	horreur
Paraskavedekatriaphobie	+17	V	V	V	F	V	F	F	drame
Rat nommé Alice, Un	TP	F	F	F	F	F	V	F	familial
Fin de ligne	+17	V	F	F	V	V	F	V	drame
Choses qui brillent, Les	AP	V	F	F	F	F	F	F	drame
Reprends-le	+17	F	V	F	F	F	F	F	comédie
Appât pour requin	TP	F	F	F	F	F	V	F	familial
Pirate en colère	AP	F	V	F	F	F	F	V	comédie
Planète potentiellement habitable	AP	F	V	F	F	V	F	F	comedie

Comme les valeurs V/F de chaque titre de film sont parcourues par l'instruction CASE, le SGBDR cherche le premier 'V' pour décider du genre de chaque film.

Voici ce qui se passe quand le code parcourt 'Grande aventure' :

```
UPDATE table_films
SET genre =
CASE
  WHEN drame = 'V' THEN 'drame'
  WHEN comedie = 'V' THEN 'comédie'
  WHEN action = 'V' THEN 'action'
  WHEN gore = 'V' THEN 'horreur'
  WHEN SF = 'V' THEN 'SF'
  WHEN pour_enfants = 'V' THEN 'familial'
  WHEN dessin_anime = 'V' THEN 'familial'
  ELSE 'divers'
END;
```

FAUX : pas encore de genre

FAUX : pas encore de genre

FAUX : pas encore de genre

FAUX : pas encore de genre

FAUX : pas encore de genre

FAUX : pas encore de genre

VRAI : genre défini comme 'familial' et nous sautons au END pour sortir du code.

Prenons-en un avec plusieurs correspondances. Là encore, nous cherchons la première valeur 'V' pour décider du genre.

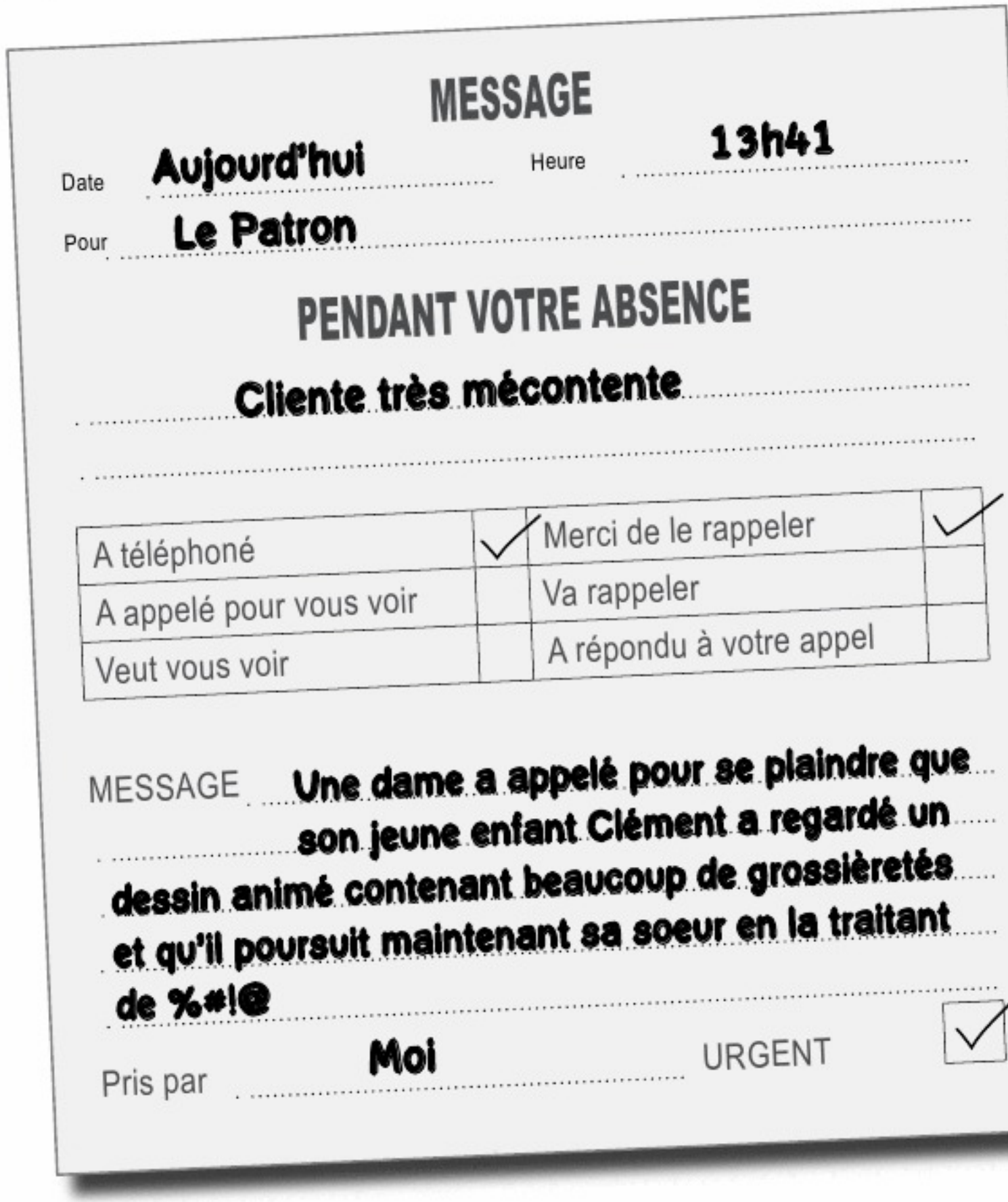
Voici ce qui se passe quand le code parcourt 'Paraskavedekatriaphobie' :

```
UPDATE table_films
SET genre =
CASE
  WHEN drame = 'V' THEN 'drame'
  WHEN comedie = 'V' THEN 'comédie'
  WHEN action = 'V' THEN 'action'
  WHEN gore = 'V' THEN 'horreur'
  WHEN SF = 'V' THEN 'SF'
  WHEN pour_enfants = 'V' THEN 'familial'
  WHEN dessin_anime = 'V' THEN 'familial'
  ELSE 'divers'
END;
```

VRAI : genre defini comme drame ; nous sautons au END et sortons du code. Toutes nos autres valeurs V sont ignorées.

On dirait que nous avons un problème

On pourrait avoir un problème. 'Grande aventure' est un dessin animé classé +17, interdit aux moins de 17 ans. D'une façon ou d'une autre, il a été classé comme 'familial'.



MESSAGE

Date Aujourd'hui Heure 13h41

Pour Le Patron

PENDANT VOTRE ABSENCE

Cliente très mécontente

A téléphoné	✓	Merci de le rappeler	✓
A appelé pour vous voir		Va rappeler	
Veut vous voir		A répondu à votre appel	

MESSAGE Une dame a appelé pour se plaindre que son jeune enfant Clément a regardé un dessin animé contenant beaucoup de grossièretés et qu'il poursuit maintenant sa soeur en la traitant de %#!@

Pris par Moi URGENT ✓

À vos crayons !

Changez l'expression CASE pour que ce dessin animé soit placé dans le genre 'divers' et non pas 'familial'.

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

MUSCLEZ VOS NEURONES

Comment pouvons-nous utiliser la classification +17 pour empêcher ce genre de choses de se reproduire ?

À vos crayons ! Solution

Changez l'expression CASE pour que ce dessin animé soit placé dans le genre 'divers' et non pas 'familial'.

```
UPDATE table_films
SET genre =
CASE
  WHEN drame = 'V' THEN 'drame'
  WHEN comedie = 'V' THEN 'comédie'
  WHEN action = 'V' THEN 'action'
  WHEN gore = 'V' THEN 'horreur'
  WHEN SF = 'V' THEN 'SF'
  WHEN pour_enfants = 'V' THEN 'familial'
  WHEN dessin_anime = 'V' AND public = 'TP' THEN 'familial'
  ELSE 'divers'
END;
```

Votre condition peut avoir plusieurs parties : ajoutez un AND à votre WHEN pour tester si le film est un dessin animé ET est classé tout public, 'TP'. Si c'est le cas, il reçoit pour genre 'familial'.

il n'y a pas de Questions stupides

Q: Suis-je obligé d'utiliser ELSE ?

R: C'est en option. Vous pouvez simplement laisser cette ligne de côté si vous n'en avez pas besoin, mais c'est sympa de pouvoir mettre à jour la valeur de votre colonne quand rien d'autre ne correspond. Il vaut mieux avoir une valeur quelconque plutôt que NULL, par exemple.

Q: Que se passe-t-il si je laisse ELSE de côté mais qu'aucune des conditions du WHEN ne correspond ?

R: Aucune valeur ne sera changée dans la colonne que vous mettez à jour.

Q: Et si je veux utiliser l'expression CASE pour certaines colonnes mais pas d'autres ? Par exemple, si je veux écrire un CASE où genre = 'divers'. Puis-je utiliser un WHERE ?

R: Oui, vous pouvez ajouter une clause WHERE après le mot-clé END. Le CASE ne s'appliquera qu'aux colonnes qui correspondent au WHERE.

Q: Puis-je utiliser une expression CASE avec autre chose que des instructions UPDATE ?

R: Oui. Vous pouvez utiliser une expression CASE avec SELECT, INSERT, DELETE et, comme vous l'avez vu, UPDATE.

CONSTRUCTION DE CASE

Votre patron, ascendant girouette, a décidé de changer un peu les choses. Lisez son email et écrivez une instruction SQL unique pour réaliser ce qu'il veut.

À : Employés de Dataville Vidéo
De : Le Patron
Sujet : De nouvelles sections impliquent de nouvelles catégories !

Ma chère famille de la vidéo,

J'ai décidé de créer de nouvelles sections. Je pense que les films interdits aux moins de 17 ans doivent être rangés dans une section différente des TP (tout public) et des AP (autorisation parentale). Créons donc 5 nouveaux genres :

horreur +17
action +17
drame +17
comedie +17
SF +17

Et s'il y a des films TP dans la section divers, mettez-les dans le rayon Familial.

Merci. Ce sera génial,
Votre patron

Il apparaît que les nouveaux genres posent des difficultés aux clients pour trouver des films. Écrivez une instruction pour vous débarrasser des nouveaux genres +17 que vous venez de créer.

Enfin, supprimez toutes ces colonnes V/F dont nous n'avons plus besoin.

SOLUTION DE CONSTRUCTION DE CASE

Votre patron, ascendant girouette, a décidé de changer un peu tout ça. Lisez son email et écrivez une instruction SQL unique pour réaliser ce qu'il veut.

```
UPDATE table_films
SET genre =
CASE
  WHEN drame = 'V' AND public = '+17' THEN 'drame +17'
  WHEN comedie = 'V' AND public = '+17' THEN 'comédie +17'
  WHEN action = 'V' AND public = '+17' THEN 'action +17'
  WHEN gore = 'V' AND public = '+17' THEN 'horreur +17'
  WHEN SF = 'V' AND public = '+17' THEN 'SF +17'
  WHEN genre = 'divers' AND public = 'TP' THEN 'familial'
END;
```

À : Employés de Dataville Video
De : Le Patron
Sujet : De nouvelles sections impliquent de nouvelles catégories !

Ma chère famille de la vidéo,

J'ai décidé de créer de nouvelles sections. Je pense que les films interdits aux moins de 17 ans doivent être rangés dans une section différente des TP (tout public) et des AP (autorisation parentale). Créons donc 5 nouveaux genres :

horreur +17
action +17
drame +17
comédie +17
SF +17

Et s'il y a des films TP dans la section divers, mettez-les dans le rayon Familial.

Merci. Ce sera génial,
Votre patron

Il apparaît que les nouveaux genres posent des difficultés aux clients pour trouver des films. Écrivez une instruction pour vous débarrasser des nouveaux genres +17 que vous venez de créer.

```
UPDATE table_films
SET genre =
CASE
  WHEN genre = 'drame +17' THEN 'drame'
  WHEN genre = 'comédie +17' THEN 'comédie'
  WHEN genre = 'action +17' THEN 'action'
  WHEN genre = 'horreur +17' THEN 'horreur'
  WHEN genre = 'SF +17' THEN 'SF'
END;
```

Enfin, supprimez toutes ces colonnes V/F dont nous n'avons plus besoin.

```
ALTER TABLE table_films
DROP COLUMN drame,
DROP COLUMN comedie,
DROP COLUMN action,
DROP COLUMN gore,
DROP COLUMN SF,
DROP COLUMN pour_enfants,
DROP COLUMN dessin_anime;
```

Les tables peuvent se transformer en fouillis

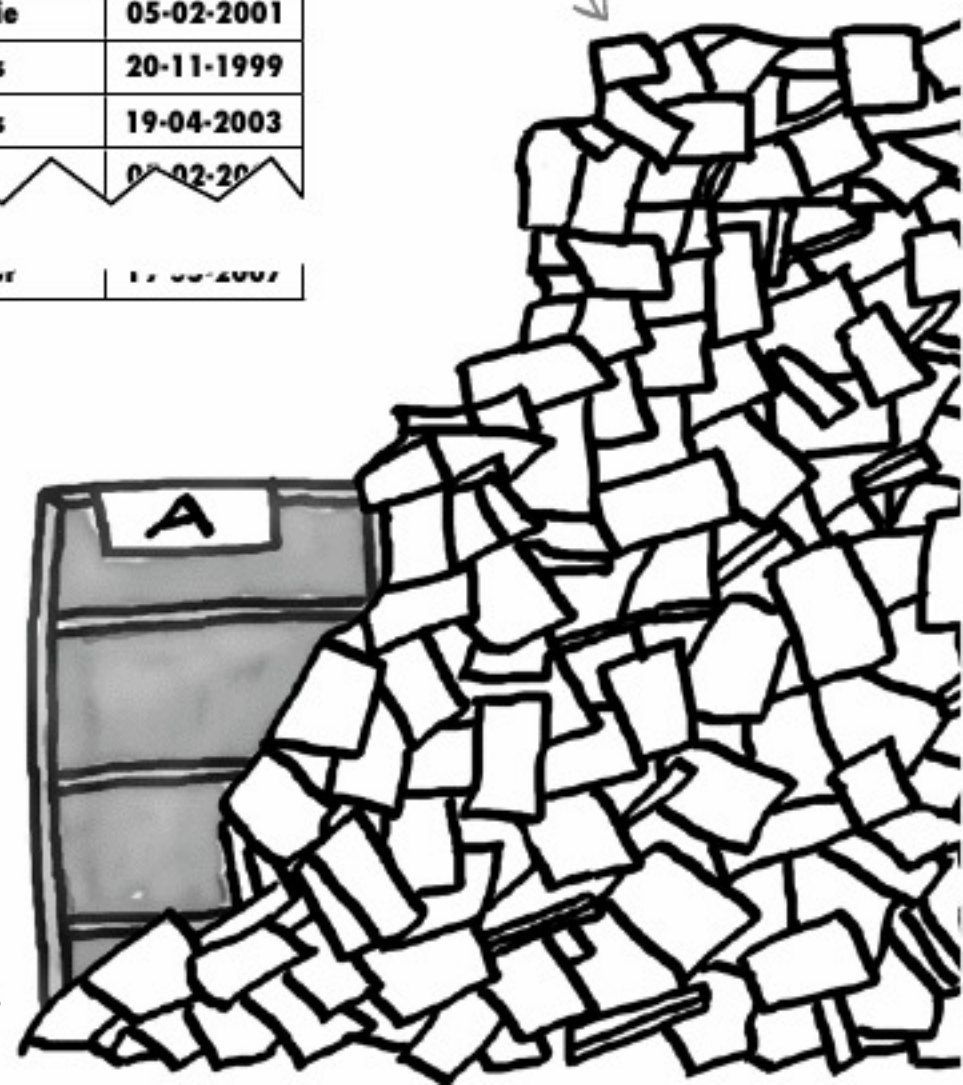
Quand un film arrive au magasin, il est ajouté à notre table et devient la ligne la plus récente de la table. Il n'y a pas d'ordre pour les films de notre table. Et maintenant qu'il est temps de remettre nos films en rayon, nous avons un petit problème. Nous savons que chacune des nouvelles étagères peut contenir 20 films et chacun des plus de 3 000 films doit avoir une étiquette indiquant son genre. ***Nous devons sélectionner les films de chaque genre, dans l'ordre alphabétique à l'intérieur du genre.***

Nous savons comment interroger la base de données pour trouver tous les films de chaque genre, mais, d'une façon ou d'une autre, nous devons les lister dans l'ordre alphabétique au sein de leur genre.

table_films

id_film	titre	public	genre	achat
				03
83	Grande aventure	TP	familial	06-03--2002
84	Greg : histoire secrète	AP	action	05-02-2001
85	Clowns fous, Les	+17	horreur	20-11-1999
86	Paraskavedekatriaphobie	+17	action	19-04-2003
87	Rat nommé Alice, Un	TP	familial	19-04-2003
88	Fin de ligne	+17	divers	05-02-2001
89	Choses qui brillent, Les	AP	drame	06-03-2002
90	Reprends-le	+17	comédie	05-02-2001
91	Appât pour requin	TP	divers	20-11-1999
92	Pirate en colère	AP	divers	19-04-2003
93	Planète potentiellement	AP	SF	05-02-20
94	Vaches en folie	+17	horreur	17-03-2007

Voici quelques-uns des plus de 3 000 films que Dataville Vidéo a en magasin.



GYM DU CERVEAU

Comment organiseriez-vous ces données dans l'ordre alphabétique en utilisant une instruction SQL ?

Il nous faut un moyen d'organiser les données SELECTionnées

Chacun des plus de 3 000 films doit porter une étiquette indiquant son genre. Puis il doit être mis en rayon dans l'ordre alphabétique.

Nous avons besoin d'une liste principale des films dans l'ordre alphabétique pour chaque genre. Pour l'instant, nous savons comment utiliser `SELECT`. Nous pouvons facilement sélectionner des films par genre et nous pouvons même les sélectionner selon le genre et la première lettre du titre.

Mais pour organiser notre longue liste de films, nous aurions besoin d'écrire au moins 182 instructions `SELECT` : en voici juste quelques-unes :

```
SELECT titre, genre FROM table_films WHERE titre LIKE 'A%' AND genre = 'familial';
SELECT titre, genre FROM table_films WHERE titre LIKE 'B%' AND genre = 'familial';
SELECT titre, genre FROM table_films WHERE titre LIKE 'C%' AND genre = 'familial';
SELECT titre, genre FROM table_films WHERE titre LIKE 'D%' AND genre = 'familial';
SELECT titre, genre FROM table_films WHERE titre LIKE 'E%' AND genre = 'familial';
SELECT titre, genre FROM table_films WHERE titre LIKE 'F%' AND genre = 'familial';
SELECT titre, genre FROM table_films WHERE titre LIKE 'G%' AND genre = 'familial';
```

Nous avons besoin de connaître le titre pour pouvoir le chercher dans la pile ainsi que le genre pour l'étiquette et la mise en rayon.

C'est la lettre de l'alphabet par laquelle les titres de films doivent commencer.

Et voici le genre que nous cherchons.

Cela fait 182 requêtes parce que nous avons 7 catégories et 26 lettres dans l'alphabet. Ce calcul n'inclut pas les films qui commencent par un nombre (comme '101 Dalmatiens, Les' ou '2001 : L'odyssée de l'espace').

MUSCLEZ VOS NEURONES

À votre avis, où vont apparaître les titres qui commencent par un nombre ou un caractère autre qu'une lettre, comme un point d'exclamation ?

À vos crayons !

Nous devons encore mettre les titres dans l'ordre alphabétique manuellement dans la liste de leur genre en utilisant les lettres qui suivent l'initiale 'A' pour décider de l'ordre.

Regardez de plus près le résultat d'une seule de nos 182 requêtes (voire plus). Essayez de classer la liste des films dans l'ordre alphabétique à la main.

```
SELECT titre, genre FROM table_films WHERE titre LIKE 'A%' AND genre = 'familial';
```

Quelques-uns de nos résultats de requête

titre	genre
Avions et hélicoptères	familial
Alors vous m'écoutez ?	familial
Art de la comédie	familial
Ah, c'est vous ma mère ?	familial
Arthur soupire	familial
Après le départ des clowns	familial
Art pour les enfants, L'	familial
Aventure animale	familial
Animaux et biscuits	familial
Autre marche des pinguins, Une	familial
Alors on grandit	familial
Aaargh!	familial
Aardvarks en folie, Les	familial
Alaska : le pays du saumon	familial
Anges	familial
Anne mange des vers	familial
Aventure extraordinaire	familial
Adultes ennuyeux	familial
Alex a besoin d'un bain	familial
Aaargh! 2	familial

À vos crayons ! Solution

Nous devons encore mettre les titres dans l'ordre alphabétique manuellement dans la liste de leur genre en utilisant les lettres qui suivent l'initiale 'A' pour décider de l'ordre.

Regardez de plus près le résultat d'une seule de nos 182 requêtes (voire plus). Essayez de classer la liste des films dans l'ordre alphabétique à la main.

```
SELECT titre, genre FROM table_films WHERE titre LIKE 'A%' AND genre = 'familial';
```

titre	genre
Aaargh !	familial
Aaargh ! 2	familial
Aardvarks en folie, Les	familial
Adultes ennuyeux	familial
Ah, c'est vous ma mère ?	familial
Alaska : le pays du saumon	familial
Alex a besoin d'un bain	familial
Alors on grandit	familial
Alors vous m'écoutez ?	familial
Anges	familial
Animaux et biscuits	familial
Anne mange des vers	familial
Après le départ des clowns	familial
Arthur soupire	familial
Art de la comédie	familial
Art pour les enfants, L'	familial
Autre marche des pinguins, Une	familial
Aventure animale	familial
Aventure extraordinaire	familial
Avions et hélicoptères	familial

Combien de temps cela vous a-t-il pris de classer ces 20 films ?

Pouvez-vous imaginer le temps que cela prendrait de classer 3 000 films ou plus de cette façon ?

Les titres commençant par 'Aventure...' viennent à la fin puisque la lettre suivant l'initiale 'A' est un 'v', mais ensuite il faut regarder la neuvième lettre du titre pour pouvoir décider où placer chaque film.

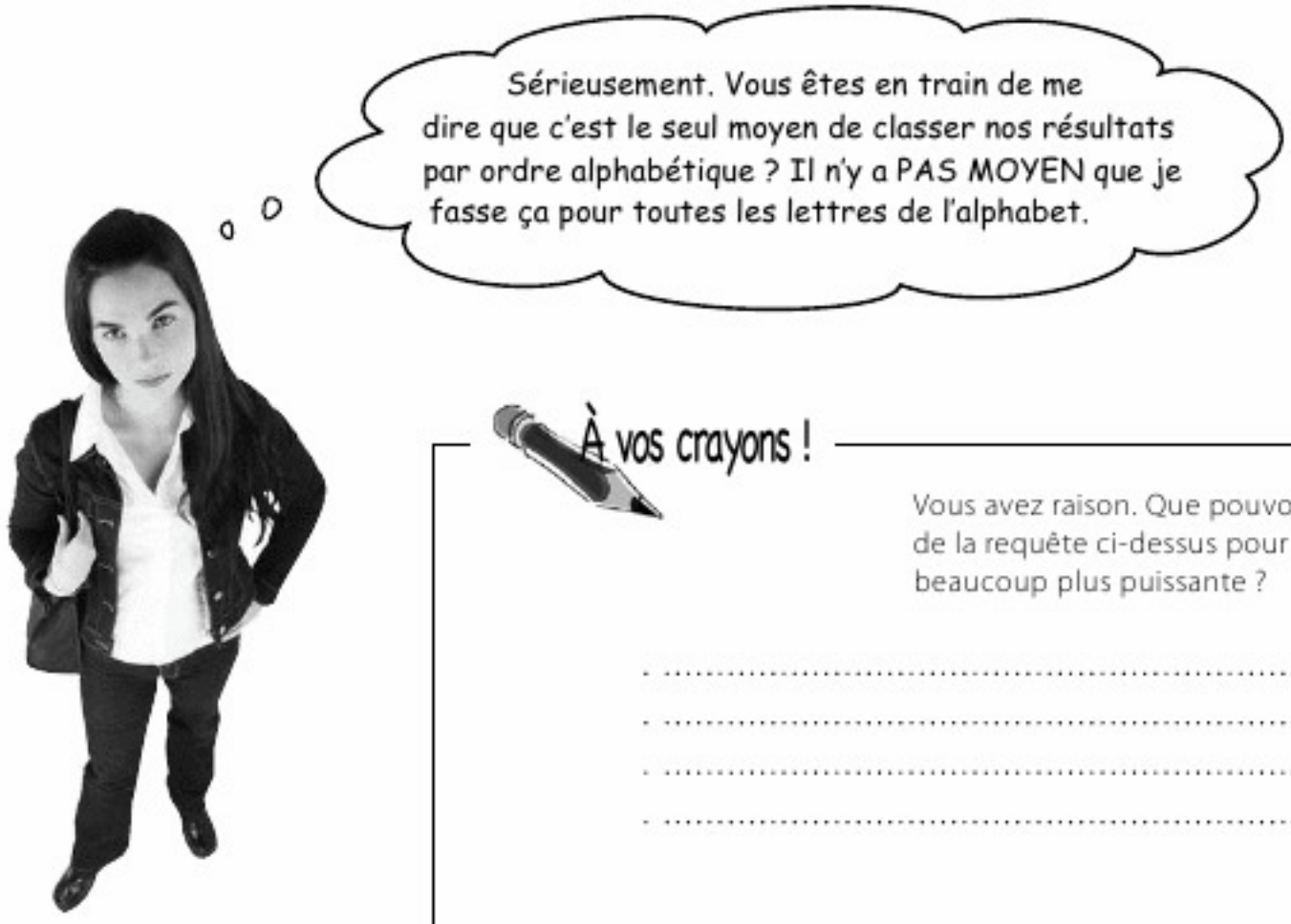
Essayez un petit ORDER BY

Vous dites que vous avez besoin de classer vos requêtes ? Hé bien, il se trouve justement que vous pouvez demander à SQL de `SELECTionner` quelque chose et de classer les données renvoyées selon une autre colonne de la table avec `ORDER BY`.

Pas de surprise dans cette partie. C'est exactement la même requête SELECT que celle que nous venons d'essayer.

```
SELECT titre, genre
FROM table_films
WHERE
titre LIKE 'A%'
AND
genre = 'familial'
ORDER BY titre;
```

Voici la nouveauté. Cela demande au programme de renvoyer les données dans l'ordre alphabétique des titres.



À vos crayons !

Vous avez raison. Que pouvons-nous enlever de la requête ci-dessus pour la rendre beaucoup plus puissante ?

. .....

. .....

. .....

. .....

STOP! Faites cet exercice avant de tourner la page.

ORDER dans une seule colonne

Si notre requête utilise `ORDER BY titre`, nous n'avons plus besoin de chercher les titres qui commencent par une lettre donnée puisque la requête renvoie les données dans l'ordre alphabétique des titres.

Nous n'avons qu'à enlever la partie `titre LIKE` et `ORDER BY titre` fera le reste.

À vos crayons ! Solution

Vous avez raison. Que pouvons-nous enlever de la requête ci-dessus pour la rendre beaucoup plus puissante ?

```
SELECT titre, genre
FROM table_films
WHERE
titre LIKE 'A%'
AND
genre = 'familial'
ORDER BY titre;
```

```
SELECT titre, genre
FROM table_films
WHERE
genre = 'familial'
ORDER BY titre;
```

Cette fois nous aurons toute la liste des films du genre familial.

Mieux encore, cette liste inclut les films dont le titre commence par un nombre. Ils sont les premiers sur la liste.

Ce n'est pas la fin des résultats ; nous n'avons pas la place de tous les montrer ici. Ils continuent jusqu'aux titres commençant par Z

ORDER BY vous permet de classer une colonne par ordre alphabétique.

Remarquez que les premiers titres commencent par un nombre.

titre	genre
1 extra-terrestre fou	familial
10 gros insectes	familial
101 Alsaciens	familial
13ème anniversaire magique	familial
2 + 2 font 5	familial
3001 façons de tomber	familial
5ème : les filles sont méchantes	familial
7 ans de tics	familial
8 bras valent mieux que 2	familial
Aaargh !	familial
Aaargh ! 2	familial
Aardvarks en folie, Les	familial
Adultes ennuyeux	familial
Ah, c'est vous ma mère ?	familial
Alaska : le pays du saumon	familial
Alex a besoin d'un bain	familial
Alors on grandit	familial
Alors vous m'écoutez ?	familial
Anges	familial
Animaux et biscuits	familial
Anne mange des vers	familial
Après le départ des clowns	familial
Arthur soupire	familial
Art de la comédie	familial
Art pour les enfants, L'	familial
Autre marche des pinguins, Une	familial
Aventure animale	familial
Aventure extraordinaire	familial
Avions et hélicoptères	familial

Créez une table simple avec une seule colonne CHAR(1) appelée 'test_caracteres'.

Insérez les chiffres, lettres (les majuscules et les minuscules) et les caractères non alphabétiques ci-dessous dans cette colonne, chacun sur une ligne séparée. Insérez un espace et laissez une ligne en NULL.

Essayez votre nouvelle requête ORDER BY sur la colonne et complétez les espaces vides dans le livre *Les règles SQL de l'ordre* ci-dessous.

```
0123ABCDabcd!@#$%^&*()-_
+=[]{};:'"\|`~,.<>/?
```

Les règles SQL de l'ordre

Après l'exécution de votre requête `ORDER BY`, complétez les blancs selon l'ordre dans lequel les caractères apparaissent dans vos résultats.

Les caractères non-alphabétiques apparaissent ..... les nombres.

Les nombres apparaissent ..... les caractères de texte.

Les valeurs NULL apparaissent ..... les nombres.

Les valeurs NULL apparaissent ..... les caractères alphabétiques.

Les caractères en majuscules apparaissent ..... les caractères en minuscules.

"A 1" apparaîtra ..... "A1".

Les règles SQL de l'ordre

Après l'exécution de votre requête `ORDER BY`, placez ces caractères dans l'ordre où ils apparaissent.

```
+ = ! ( & ~ "
* @ ? '
```

Vous souvenez-vous comment insérer une apostrophe ? Il y a une ruse.

Créez une table simple avec une seule colonne CHAR(1) appelée 'test_caracteres'.

Insérez les chiffres, lettres (les majuscules et les minuscules) et les caractères non alphabétiques ci-dessous dans cette colonne, chacun sur une ligne séparée. Insérez un espace et laissez une ligne en NULL.

Essayez votre nouvelle requête ORDER BY sur la colonne et complétez les espaces vides dans le livre *Les règles SQL de l'ordre* ci-dessous.

```
 !"#$%&'()*+,-./0123:;<=>
?@ABCD[\]^_`abcd{|}~
```

Voici l'ordre dans lequel les caractères ont pu s'afficher dans vos résultats. Notez que l'espace est au début. Votre ordre peut être légèrement différent selon votre SGBDR. L'important est de savoir qu'il EXISTE un ordre et quel est cet ordre pour votre SGBDR.

Les règles SQL de l'ordre

Après l'exécution de votre requête ORDER BY, complétez les blancs selon l'ordre dans lequel les caractères apparaissent dans vos résultats.

Les caractères non-alphabétiques apparaissent avant et après les nombres.

Les nombres apparaissent avant les caractères de texte.

Les valeurs NULL apparaissent avant les nombres.

Les valeurs NULL apparaissent avant les caractères alphabétiques.

Les caractères en majuscules apparaissent avant les caractères en minuscules.

"A 1" apparaîtra avant "A1".

Les règles SQL de l'ordre

Après l'exécution de votre requête ORDER BY, placez ces caractères dans l'ordre où ils apparaissent.

+ = ! (& ~ "
* @ ? '

! " & ' (* + = ? @ ~

ORDER avec deux colonnes

Il semble que tout est sous contrôle. Nous pouvons classer nos films par ordre alphabétique et nous pouvons créer des listes alphabétiques pour chaque catégorie.

Malheureusement, votre patron a quelque chose d'autre à vous donner à faire…

À : Employés de Dataville Video
De : Le Patron
Sujet : On vire les vieux (films)

Salut,

Je pense que nous devons nous débarrasser de certains des films que nous avons depuis trop longtemps. Pouvez-vous venir ce week-end me donner une liste des films de chaque genre classés par date d'achat ?

Ce serait super,
Votre patron

Heureusement, vous pouvez classer plusieurs colonnes dans la même instruction.

Nous voulons être sûr que la date d'achat apparaisse dans les résultats.

```
SELECT titre, genre, achat
FROM table_films
ORDER BY genre, achat;
```

Ce sera la première colonne classée. Nous aurons une liste de tous les films du magasin, classés par genre.

Et ce sera la deuxième colonne classée, APRÈS que la colonne genre ait été classée.

GYM DU CERVEAU

Les films les plus vieux seront-ils les premiers ou les derniers de chaque genre ? Et que pensez-vous qu'il se passera si deux films du même genre ont la même date d'achat ? Lequel apparaîtra en premier ?

ORDER avec plusieurs colonnes

Vous n'êtes pas limité à un tri sur deux colonnes. Vous pouvez classer selon autant de colonnes que nécessaire pour obtenir les données que vous voulez.

Regardez cet ORDER BY avec trois colonnes. Voici ce qui se passe et la façon dont la table est triée.

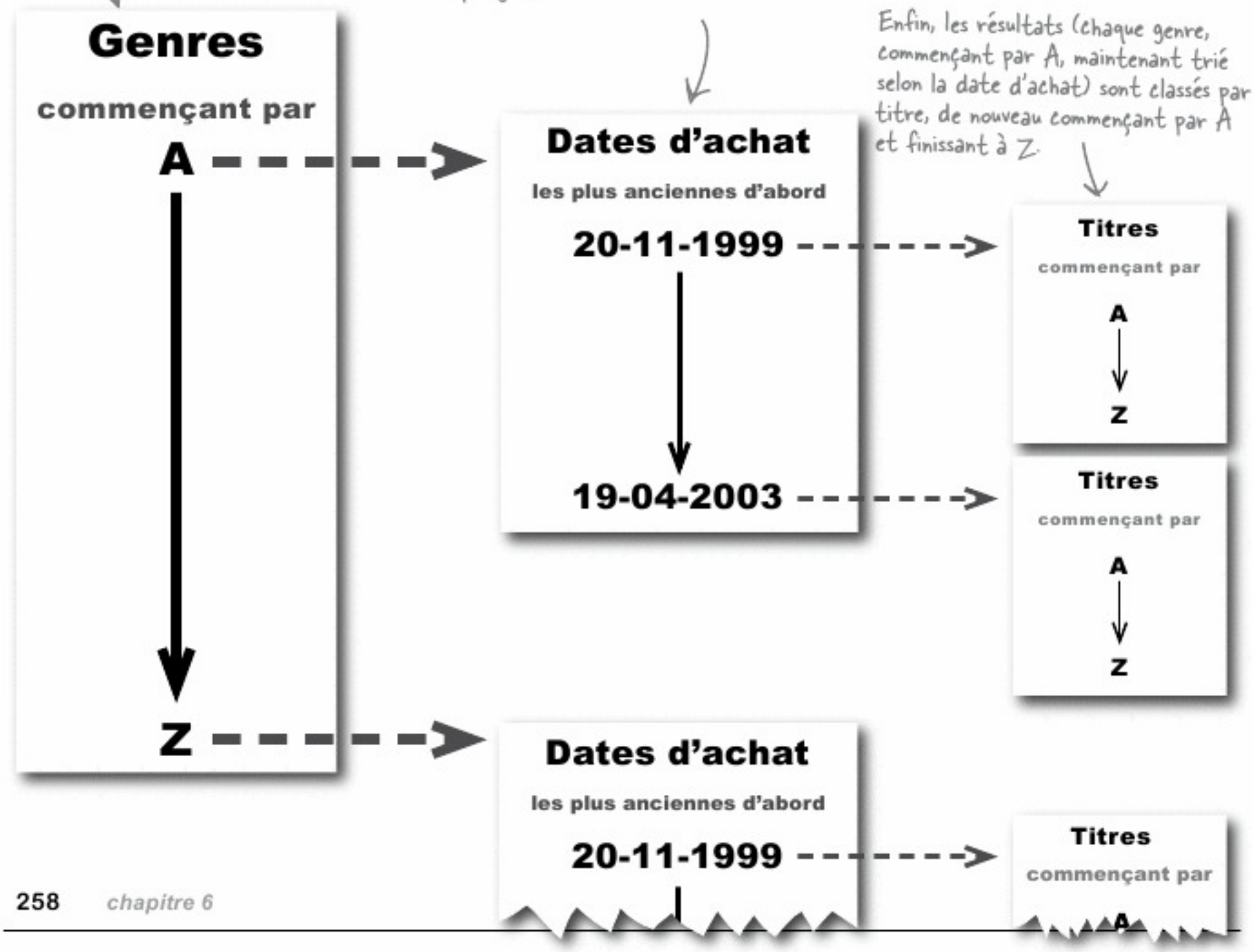
Vous pouvez trier selon autant de colonnes que nécessaire.

```
SELECT * FROM table_films
ORDER BY genre, achat, titre;
```

Vos résultats sont d'abord classés par genre, puisque c'est la première colonne listée après ORDER BY. Les résultats sont listés de A à Z.

Ensuite, les résultats (chaque genre de la table commençant par A puisque c'est maintenant ainsi que les genres sont classés) sont triés par date, avec la plus ancienne date en premier. Les dates sont toujours triées par année, puis par mois, puis par jour.

Enfin, les résultats (chaque genre, commençant par A, maintenant trié selon la date d'achat) sont classés par titre, de nouveau commençant par A et finissant à Z.



Une table_films ordonnée

Voyons ce que cette instruction SELECT renvoie vraiment si nous la lançons sur notre table de films d'origine.

Notre table_films d'origine

Il n'y a pas réellement d'ordre ici ; les films s'affichent dans l'ordre dans lequel les enregistrements ont été insérés dans la table.

id_film	titre	public	genre	achat
83	Aventure de Roger, L'	TP	familial	06-03-2002
84	Greg : histoire secrète	AP	action	05-02-2001
85	Clowns fous, Les	+17	horreur	20-11-1999
86	Paraskavedekatriaphobie	+17	action	19-04-2003
87	Rat nommé Alice, Un	TP	familial	19-04-2003
88	Fin de ligne	+17	divers	05-02-2001
89	Choses qui brillent, Les	AP	drame	06-03-2002
90	Reprends-le	+17	comedie	05-02-2001
91	Appât pour requin	TP	divers	20-11-1999
92	Pirate en colère	AP	divers	19-04-2003
93	Planète potentiellement habitable	AP	SF	05-02-2001
9[illegible]	Vaches en folie	+17	horreur	3[illegible]-9-200[illegible]

et les résultats triés par notre requête :

Troisième colonne triée

Première colonne à avoir été triée

Deuxième colonne triée

id_film	titre	public	genre	achat
84	Greg : histoire secrète	AP	action	05-02-2001
86	Paraskavedekatriaphobie	+17	action	19-04-2003
90	Reprends-le	+17	comedie	05-02-2001
91	Appât pour requin	TP	divers	20-11-1999
88	Fin de ligne	+17	divers	05-02-2001
89	Choses qui brillent, Les	AP	drame	06-03-2002
83	Aventure de Roger, L'	TP	familial	06-03-2002
87	Rat nommé Alice, Un	TP	familial	19-04-2003
85	Clowns fous, Les	+17	horreur	20-11-1999
93	Planète potentiellement	AP	SF	05-02-2001

SQL a un mot-clé qui inverse l'ordre.

Par défaut, SQL renvoie vos colonnes ordonnées avec `ORDER BY` dans l'ordre `ASCENDANT`. Ce qui signifie que vous obtenez toujours A vers Z et 1 vers 99 999. Si vous préférez que l'ordre soit inversé, vous voulez les données dans l'ordre descendant. Vous pouvez utiliser le mot-clé `DESC` juste après le nom de la colonne.

il n'y a pas de
Questions stupides

Q: Je croyais qu'on utilisait DESC pour obtenir la DESCRIPTION d'une table. Êtes-vous sûr que cela fonctionne pour changer l'ordre d'ORDER ?

R: Oui. Tout dépend du contexte. Si vous l'utilisez devant un nom de table (par exemple, **`DESC table_films;`**), vous aurez une description de la table. Dans ce cas, c'est l'abréviation de **`DESCRIBE`**.

Si vous **l'utilisez dans une clause `ORDER`**, cela signifie **`DESCENDING`** et il trie les résultats dans l'ordre descendant.

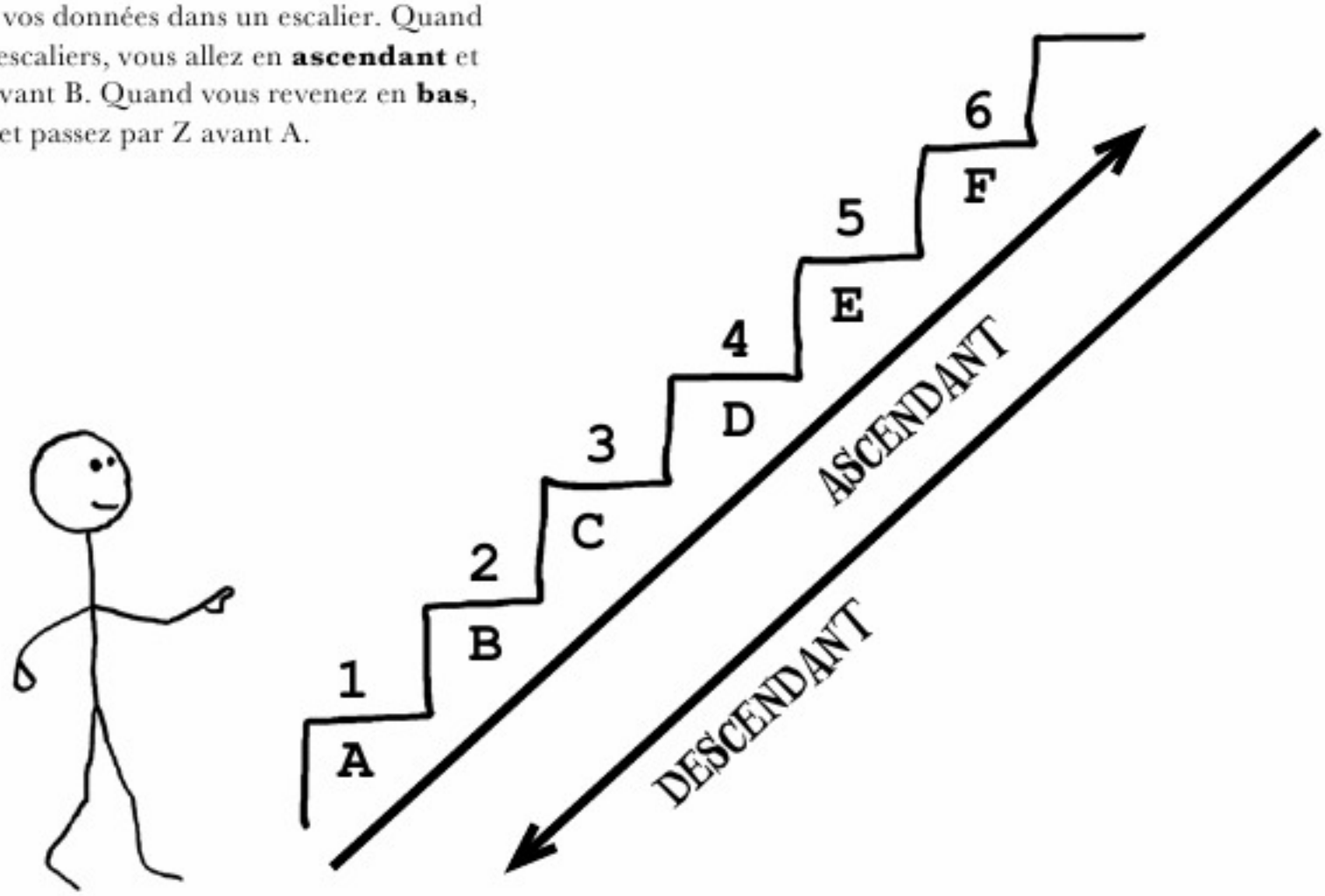
Q: Puis-je utiliser les mots DESCRIBE et DESCENDING en entier dans mes requêtes pour éviter les confusions ?

R: Vous pouvez utiliser **`DESCRIBE`**, mais `DESCENDING` ne fonctionnera pas.

Utilisez le mot-clé DESC après le nom de votre colonne dans les clauses ORDER BY pour inverser l'ordre de vos résultats.

Inverser l'ORDER avec DESC

Représentez-vous vos données dans un escalier. Quand vous **montez** les escaliers, vous allez en **ascendant** et vous arrivez à A avant B. Quand vous revenez en **bas**, vous **descendez** et passez par Z avant A.



Cette requête nous donne une liste de films triés par date d'achat avec les plus ***récents*** en premier. Pour chaque date, les films achetés sont listés dans l'ordre alphabétique.

```
SELECT titre, achat
FROM table_films
ORDER BY titre ASC, achat DESC;
```

Nous pouvons mettre ASC ici mais ce n'est pas nécessaire. Souvenez-vous simplement que ASC est l'ordre par défaut.

Si nous voulons trier nos données de Z à A ou de 9 à 1, nous devons utiliser le mot-clé DESC.

À : Employés de Dataville Vidéo
De : Le Patron
Sujet : Cadeau pour tous !

Salut,

Le magasin a beaucoup d'allure ! Tous les films sont empilés au bon endroit et, grâce à ces sympathiques clauses ORDER BY dans votre SQL, tout le monde peut trouver exactement ce qu'il cherche.

Pour vous remercier de votre dur labeur, j'organise une petite soirée pizza chez moi ce soir. Venez vers 18h.

N'oubliez pas d'apporter ces rapports !
Votre patron

P.S. Ne mettez pas de vêtements trop chics, j'ai des étagères de livres que j'aimerais bien réorganiser…

Le problème de la cheftaine des Jeannettes

La cheftaine de la troupe locale des Jeannettes essaie de trouver quelle fille a vendu le plus de biscuits. Jusqu'ici elle a une table des ventes de chaque fille pour chaque jour.

Jeannette ayant réalisé la vente

Montant gagné en euros

Ventes pour ce jour

ventes_biscuits

ID	prenom	ventes	date_vente
1	Lindsay	32.02	06-03-2007
2	Paris	26.53	06-03-2007
3	Britney	11.25	06-03-2007
4	Nicole	18.96	06-03-2007
5	Lindsay	9.16	07-03-2007
6	Paris	1.52	07-03-2007
7	Britney	43.21	07-03-2007
8	Nicole	8.05	07-03-2007
9	Lindsay	17.62	08-03-2007
10	Paris	24.19	08-03-2007
11	Britney	3.40	08-03-2007
12	Nicole	15.21	08-03-2007
13	Lindsay	0	09-03-2007
14	Paris	31.99	09-03-2007
15	Britney	2.58	09-03-2007
16	Nicole	0	09-03-2007
17	Lindsay	2.34	10-03-2007
18	Paris	13.44	10-03-2007
19	Britney	8.78	10-03-2007
20	Nicole	26.82	10-03-2007
21	Lindsay	3.71	11-03-2007
22	Paris	.56	11-03-2007
23	Britney	34.19	11-03-2007
24	Nicole	7.77	11-03-2007
25	Lindsay	16.23	12-03-2007
26	Paris	0	12-03-2007
27	Britney	4.50	12-03-2007
28	Nicole	19.22	12-03-2007

À vos crayons !

La Jeannette avec le plus gros montant vendu va gagner des leçons d'équitation gratuites. Toutes les Jeannettes veulent gagner, il est donc crucial qu'Edwina trouve la vraie gagnante avant que les choses tournent au massacre.

Utilisez vos nouvelles compétences d'ORDER BY pour écrire une requête qui aidera Edwina à trouver le nom de la gagnante.

À vos crayons ! Solution

La Jeannette avec le plus gros montant vendu va gagner des leçons d'équitation gratuites. Toutes les Jeannettes veulent gagner, il est donc crucial qu'Edwina trouve la vraie gagnante avant que les choses tournent au massacre.

Utilisez vos nouvelles compétences d'ORDER BY pour écrire une requête qui aidera Edwina à trouver le nom de la gagnante.

```
SELECT prenom, ventes
FROM ventes_biscuits
ORDER BY prenom;
```

Voici notre requête

...et voici les résultats.

prenom	ventes
Britney	3.40
Britney	2.58
Britney	4.50
Britney	11.25
Britney	8.78
Britney	43.21
Britney	34.19
Lindsay	17.62
Lindsay	9.16
Lindsay	0.00
Lindsay	32.02
Lindsay	2.34
Lindsay	3.71
Lindsay	16.23
Nicole	19.22
Nicole	0.00
Nicole	8.05
Nicole	26.82
Nicole	7.77
Nicole	15.21
Nicole	18.96
Paris	26.53
Paris	0.00
Paris	0.56
Paris	1.52
Paris	13.44
Paris	24.19
Paris	31.99

107.91

81.08

96.03

98.23

Les ventes pour chacune des filles devaient être calculées manuellement pour trouver la gagnante..

SUM peut les additionner pour nous

Les enjeux sont de taille. Nous ne pouvons pas faire d'erreur et risquer de mettre nos Jeannettes en colère. Plutôt que d'additionner tout cela nous-mêmes, nous pouvons charger SQL de faire le travail ingrat à notre place.

La langage SQL a des mots-clés spéciaux appelés *fonctions*. Les fonctions sont des morceaux de code qui réalisent une opération sur une ou plusieurs valeurs. La première que nous allons vous montrer réalise une opération mathématique sur une colonne. Nous utiliserons la fonction `SUM` qui fait le ***total des valeurs dans une colonne*** désignée entre parenthèses. Regardons-la à l'œuvre.

La fonction SUM calcule le total des valeurs de la colonne ventes.

SUM est appelée une fonction. Cela signifie qu'elle réalise une action sur la colonne qui se trouve à côté d'elle entre parenthèses.

```
SELECT SUM(ventes)
FROM ventes_biscuits
WHERE prenom = 'Nicole';
```

Ceci limite la requête à l'addition des ventes de Nicole. Sans quoi elle calculerait le total de la colonne des ventes tout entière.

```
Fichier Edition Fenêtre Aide LaGagnanteEst
> SELECT SUM(ventes) FROM ventes_biscuits
-> WHERE prenom = 'Nicole';
+------------+
| SUM(ventes)|
+------------+
|      96.03 |
+------------+
1 row in set (0.00 sec)
```

Nous avons encore besoin des trois autres totaux et nous aurons fini. Mais ce serait plus simple si nous pouvions tout faire en une seule requête...

À faire à la maison

Essayez. Créez une table comme la table ventes_biscuits et insérez-y des valeurs décimales. Puis appliquez rigoureusement les requêtes que vous trouverez dans les pages qui suivent.

SUM tout en un avec GROUP BY

Il existe un moyen de faire la somme (SUM) des ventes de chaque Jeannette en même temps. Nous allons simplement ajouter un GROUP BY à notre instruction SUM. Cela regroupe toutes les valeurs selon le prénom de chaque fille et calcule le total des ventes pour ce groupe.

```
SELECT  prenom, SUM(ventes)
FROM ventes_biscuits
GROUP BY prenom
ORDER BY SUM(ventes)DESC;
```

SUM fait la somme des nombres de la colonne des ventes.

Group regroupe toutes les valeurs par prénom.

Nous voulons que les valeurs soient affichées des plus hautes aux plus basses pour voir la gagnante plus facilement.

Nous devons trier selon le même SUM que nous avons sélectionné.

Cette instruction totalise toutes les valeurs ventes dans chaque groupe prénom.

prenom	ventes
Nicole	19.22
Nicole	0.00
Nicole	8.05
Nicole	26.82
Nicole	7.77
Nicole	15.21
Nicole	18.96

prenom	ventes
Paris	26.53
Paris	0.00
Paris	0.56
Paris	1.52
Paris	13.44
Paris	24.19
Paris	31.99

prenom	ventes
Lindsay	17.62
Lindsay	9.16
Lindsay	0.00
Lindsay	32.02
Lindsay	2.34
Lindsay	3.71
Lindsay	16.23

prenom	ventes
Britney	3.40
Britney	2.58
Britney	4.50
Britney	11.25
Britney	8.78
Britney	43.21
Britney	34.19

Et la gagnante est Britney !

```
Fichier Edition Fenêtre Aide LaGagnanteEstEnFait
> SELECT prenom, SUM(ventes)
-> FROM ventes_biscuits GROUP BY prenom
-> ORDER BY SUM(ventes) DESC;
+------------+------------+
| prenom     | sum(ventes)|
+------------+------------+
| Britney    |     107.91 |
| Paris      |      98.23 |
| Nicole     |      96.03 |
| Lindsay    |      81.08 |
+------------+------------+
4 rows in set (0.00 sec)
```

AVG avec GROUP BY

Les autres filles étaient déçues, Edwina a donc décidé d'accorder un autre prix à la fille avec la plus haute moyenne journalière. Elle utilise la fonction AVG.

Chaque fille a sept jours de ventes. Pour chacune, la fonction AVG ajoute leurs ventes puis les divise par 7.

De nouveau, nous regroupons toutes les valeurs par prénom...

... mais cette fois nous faisons la moyenne des valeurs.

```
SELECT prenom, AVG(ventes)
FROM ventes_biscuits
GROUP BY prenom;
```

AVG ajoute toutes les valeurs d'un groupe puis divise par le nombre total de valeurs pour trouver la valeur moyenne pour ce groupe.

prenom	ventes
Nicole	19.22
Nicole	0.00
Nicole	8.05
Nicole	26.82
Nicole	7.77
Nicole	15.21
Nicole	18.96

prenom	ventes
Paris	26.53
Paris	0.00
Paris	0.56
Paris	1.52
Paris	13.44
Paris	24.19
Paris	31.99

prenom	ventes
Lindsay	17.62
Lindsay	9.16
Lindsay	0.00
Lindsay	32.02
Lindsay	2.34
Lindsay	3.71
Lindsay	16.23

prenom	ventes
Britney	3.40
Britney	2.58
Britney	4.50
Britney	11.25
Britney	8.78
Britney	43.21
Britney	34.19

Oups, Britney gagne encore. Nous devons trouver un autre moyen pour déterminer la gagnante de la seconde place.

Fichier Edition Fenêtre Aide

```
> SELECT  prenom, AVG(ventes)
-> FROM ventes_biscuits GROUP BY prenom;
+------------+------------+
| prenom     | AVG(ventes)|
+------------+------------+
| Nicole     |  13.718571 |
| Britney    |  15.415714 |
| Lindsay    |  11.582857 |
| Paris      |  14.032857 |
+------------+------------+
4 rows in set (0.00 sec)
```

MIN et MAX

N'étant pas disposée à laisser quoi que ce soit de côté, Edwina jette un œil aux valeurs `MIN` et `MAX` de sa table pour voir si une des autres filles a une valeur de ventes plus grande pour un jour ou même si Britney a un mauvais jour et une valeur plus basse que toutes les autres…

Nous pouvons utiliser la fonction `MAX` pour trouver la **plus grande valeur d'une colonne**. `MIN` nous donne la **plus petite valeur d'une colonne**.

```
SELECT prenom, MAX(ventes)
FROM ventes_biscuits
GROUP BY prenom;
```

MAX renvoie la valeur de vente la plus grande pour chaque fille.

prenom	ventes
Nicole	26.82
Britney	43.21
Lindsay	32.02
Paris	31.99

Surprise, Britney a le plus gros montant de ventes par jour.

```
SELECT prenom, MIN(ventes)
FROM ventes_biscuits
GROUP BY prenom;
```

MIN renvoie la valeur de ventes la plus basse pour chaque fille.

prenom	ventes
Nicole	0.00
Britney	2.58
Lindsay	0.00
Paris	0.00

Et alors qu'on dirait que les autres filles se sont relâchées au moins une journée chacune, Britney a gagné de l'argent même pendant son plus mauvais jour.

Cela devient sérieux. Je peux peut-être donner le prix à celle qui a vendu des biscuits plus de jours que les autres.

COUNT compte les jours

Pour trouver qui a vendu des biscuits plus de jours que les autres, Edwina essaie de déterminer combien de jours les biscuits ont été vendus avec la fonction COUNT. COUNT va renvoyer le ***nombre de lignes d'une colonne***.

```
SELECT COUNT(date_vente)
FROM ventes_biscuits;
```

COUNT renvoie le nombre de lignes dans la colonne date_vente. Si la valeur est NULL, elle n'est pas comptée.

À vos crayons !

ntes_biscuits

ID	prenom	ventes	date_vente
1	Lindsay	32.02	06-03-2007
2	Paris	26.53	06-03-2007
3	Britney	11.25	06-03-2007
4	Nicole	18.96	06-03-2007
5	Lindsay	9.16	07-03-2007
6	Paris	1.52	07-03-2007
7	Britney	43.21	07-03-2007
8	Nicole	8.05	07-03-2007
9	Lindsay	17.62	08-03-2007
10	Paris	24.19	08-03-2007
11	Britney	3.40	08-03-2007
12	Nicole	15.21	08-03-2007
13	Lindsay	0	09-03-2007
14	Paris	31.99	09-03-2007
15	Britney	2.58	09-03-2007
16	Nicole	0	09-03-2007
17	Lindsay	2.34	10-03-2007
18	Paris	13.44	10-03-2007
19	Britney	8.78	10-03-2007
20	Nicole	26.82	10-03-2007
21	Lindsay	3.71	11-03-2007
22	Paris	.56	11-03-2007
23	Britney	34.19	11-03-2007
24	Nicole	7.77	11-03-2007
25	Lindsay	16.23	12-03-2007
26	Paris	0	12-03-2007
27	Britney	4.50	12-03-2007
28	Nicole	19.22	12-03-2007

Voici la table initiale. À votre avis, que va renvoyer la requête ?

Ce nombre représente-t-il le nombre réel de jours de ventes de biscuits ?

Écrivez une requête qui nous donne le nombre de jours où chaque fille a vendu des biscuits.

À vos crayons ! Solution

Voici la table initiale. À votre avis, que va renvoyer la requête ?

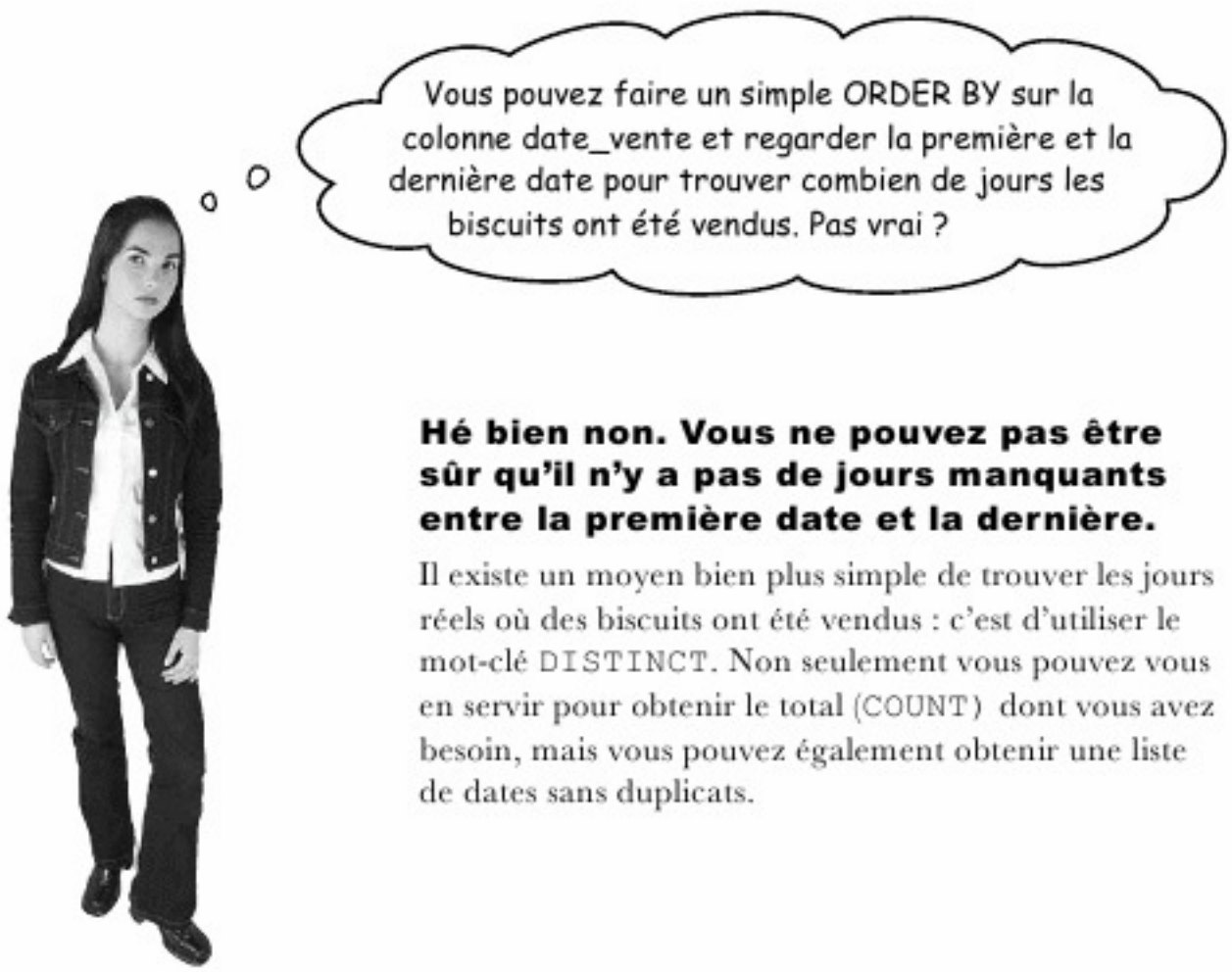
28 dates de ventes

Ce nombre représente-t-il le nombre réel de jours de ventes de biscuits ?

Non. Ce nombre représente simplement le nombre de valeurs pour date_vente dans la table.

Écrivez une requête qui nous donne le nombre de jours où chaque fille a vendu des biscuits.

```
SELECT prenom, COUNT(date_vente)
FROM ventes_biscuits
GROUP BY prenom;
```



Hé bien non. Vous ne pouvez pas être sûr qu'il n'y a pas de jours manquants entre la première date et la dernière.

Il existe un moyen bien plus simple de trouver les jours réels où des biscuits ont été vendus : c'est d'utiliser le mot-clé `DISTINCT`. Non seulement vous pouvez vous en servir pour obtenir le total (`COUNT`) dont vous avez besoin, mais vous pouvez également obtenir une liste de dates sans duplicats.

SELECT DISTINCT sélectionne des valeurs distinctes

Regardons d'abord ce mot-clé DISTINCT ***sans*** la fonction COUNT.

Comme DISTINCT est un mot-clé et non pas une fonction, vous n'avez pas besoin de parenthèses autour de date_vente.

```
SELECT DISTINCT date_vente
FROM ventes_biscuits
ORDER BY date_vente;
```

Voici notre ORDER BY pour que nous puissions voir la première et la dernière date de vente.

```
Fichier Edition Fenêtre Aide PasDeGruge
> SELECT DISTINCT date_vente
FROM ventes_biscuits
 -> ORDER BY date_vente;
+------------+
| date_vente |
+------------+
| 2007-03-06 |
| 2007-03-07 |
| 2007-03-08 |
| 2007-03-09 |
| 2007-03-10 |
| 2007-03-11 |
| 2007-03-12 |
+------------+
7 rows in set (0.00 sec)
```

Regardez ça, pas un seul duplicat !

À présent, essayons-le ***avec*** la fonction COUNT :

Remarquez que le DISTINCT va à l'intérieur des parenthèses avec date_vente.

```
SELECT COUNT(DISTINCT date_vente)
FROM ventes_biscuits;
```

Nous n'avons pas besoin d'un ORDER BY parce que COUNT ne renvoie qu'un seul nombre. Il n'y a rien à trier pour ORDER ici.

GYM DU CERVEAU

Essayez cette requête puis utilisez-la pour trouver qui a vendu des biscuits le plus de jours ?

Réponse : Britney

Qui suis-je ?

Un groupe de mots-clés et de fonctions SQL, entièrement déguisés, organisent un petit jeu, « Qui suis-je ? ». Ils vous donnent un indice et vous essayez de deviner qui ils sont à partir de ce qu'ils disent. Supposez qu'ils disent toujours la vérité. Complétez les blancs à droite pour identifier les participants. De même, pour chaque participant, notez si c'est une fonction ou un mot-clé.

Les partipants de ce soir :

COUNT, DISTINCT, AVG, MIN, GROUP BY, SUM, MAX

	Nom	fonction ou mot-clé
Le résultat que vous obtenez en m'utilisant ne vaut sans doute pas grand chose.	.....	.....
Ce que je recrache est plus gros que ce que je me mets dans la bouche.	.....	.....
Je vais vous donner des résultats absolument uniques.	.....	.....
Je vais vous dire combien ils étaient.	.....	.....
Vous devez m'utiliser si vous voulez obtenir une somme.	.....	.....
Je ne m'intéresse qu'au plus gros nombre.	.....	.....
Qui suis-je ? Quelque part au milieu.	.....	.....

Réponses page 279.

il n'y a pas de

Questions stupides

Q: Comme vous cherchiez les plus grandes valeurs avec AVG, MAX et MIN, est-ce que vous n'auriez pas pu ajouter une clause ORDER BY ?

R: Nous aurions pu et cela aurait été une très bonne idée. Nous avons choisi de le laisser de côté pour ne pas encombrer les requêtes et pour vous faciliter l'apprentissage des nouvelles fonctions. Retournez voir ces fonctions et visualisez l'ajout d'ORDER BY. Voyez-vous comment les résultats en seraient changés ?

Q: Ce mot-clé DISTINCT a l'air plutôt utile. Puis-je l'utiliser avec toutes les colonnes que je veux ?

R: Vous pouvez. Il est particulièrement utile quand vous avez de nombreux enregistrements avec la même valeur dans une colonne donnée et que vous ne voulez voir que la variété des valeurs et non pas une longue liste de valeurs dupliquées.

Q: La requête pour MIN() n'avait pas vraiment grand chose à voir avec la gagnante d'Edwina, n'est-ce pas ?

R: Non, mais cela l'a aidée à trouver les filles qui ont le moins bien travaillé. L'année prochaine, elle peut les surveiller pour les motiver davantage.

Q: En parlant de MIN, que se passe-t-il s'il y a un NULL dans la colonne ?

R: Bonne question. NULL n'est jamais renvoyé par aucune de ces fonctions parce que NULL est l'absence d'une valeur, ce n'est pas la même chose que zéro.

Musclez vos neurones

Imaginez que nous n'ayons pas quatre, mais ***quarante*** Jeannettes. Comment pourrions-nous utiliser SUM pour trouver la deuxième place ?

LIMITer le nombre de résultats

Nous allons maintenant utiliser SUM pour déterminer la seconde place. Retournons voir la requête initiale et ses résultats pour nous aider à trouver cette gagnante.

```
SELECT prenom, SUM(ventes)
FROM ventes_biscuits
GROUP BY prenom
ORDER BY SUM(ventes)DESC;
```

Il est crucial d'utiliser ORDER BY ici ; sans quoi nos résultats seraient arbitraires.

Nous ne voulons vraiment que les deux premiers résultats.

prenom	ventes
Britney	107.91
Paris	98.23
Nicole	96.03
Lindsay	81.08

Paris est notre gagnante de la seconde place ! Nicole ne lui parle plus.

Comme nous n'avons que quatre résultats, il est facile de voir qui est en deuxième place. Mais si nous voulons être encore plus précis, nous pouvons LIMITer le nombre de résultats aux deux filles de tête. Ainsi nous pouvons voir précisément les résultats que nous voulons. LIMIT nous permet de préciser exactement le nombre de lignes que nous voulons voir de notre ensemble de résultats.

```
SELECT prenom, SUM(ventes)
FROM ventes_biscuits
GROUP BY prenom
ORDER BY SUM(ventes)DESC
LIMIT 2;
```

Cela indique que vous voulez LIMITer vos résultats aux deux premiers.

C'est une longue requête et elle ne vous donne que ces deux petits résultats.

prenom	ventes
Britney	107.91
Paris	98.23

Même s'il n'y a que quatre Jeannettes dans la table et que les limiter à deux n'aide pas beaucoup ici, imaginez que vous travaillez avec une table bien plus grande. Supposez que vous ayez une liste des 1 000 meilleures chansons passant sur les radios, mais que vous ne vouliez que les 100 les plus populaires. LIMIT vous permettrait de ne voir que celles-là et pas les 900 autres chansons.

LIMITé à la seconde place

LIMIT nous permet même d'afficher la gagnante de la seconde place sans qu'on doive voir la gagnante de la première place. Pour cela, nous pouvons utiliser LIMIT avec deux paramètres :

Si vous essayiez de deviner le résultat, vous auriez probablement faux. Quand vous avez deux paramètres au lieu d'un, la signification est totalement différente.

```
LIMIT 0,4
```

prenom	ventes
Britney	107.91
Paris	98.23
Nicole	96.03
Lindsay	81.08

Britney est 0, Paris est 1, Nicole est 2 et Lindsay est 3

Voici le résultat par lequel commencer. SQL commence à compter avec 0.

Voici le nombre de résultats à renvoyer.

Vous souvenez-vous de nos 100 meilleures chansons ? Supposez que nous voulions voir les chansons de 20 à 30. Ajouter un paramètre supplémentaire à notre LIMIT nous aiderait vraiment. Nous pourrions simplement les trier par popularité, puis ajouter LIMIT 19,10. Le 19 indique de commencer à la 20ème chanson puisque SQL compte en commençant par 0 et le 10 indique de nous renvoyer 10 lignes.

À vos crayons !

Écrivez la requête qui donnera le second résultat ***et seulement le second résultat*** en utilisant la clause LIMIT avec deux paramètres.

.....

.....

.....

.....

.....

.....

.....

À vos crayons ! Solution

Écrivez la requête qui donnera le second résultat ***et seulement le second résultat*** en utilisant la clause LIMIT avec deux paramètres.

```
SELECT prenom, SUM(ventes)
FROM ventes_biscuits
GROUP BY prenom
ORDER BY SUM(ventes) DESC
LIMIT 1,1;
```

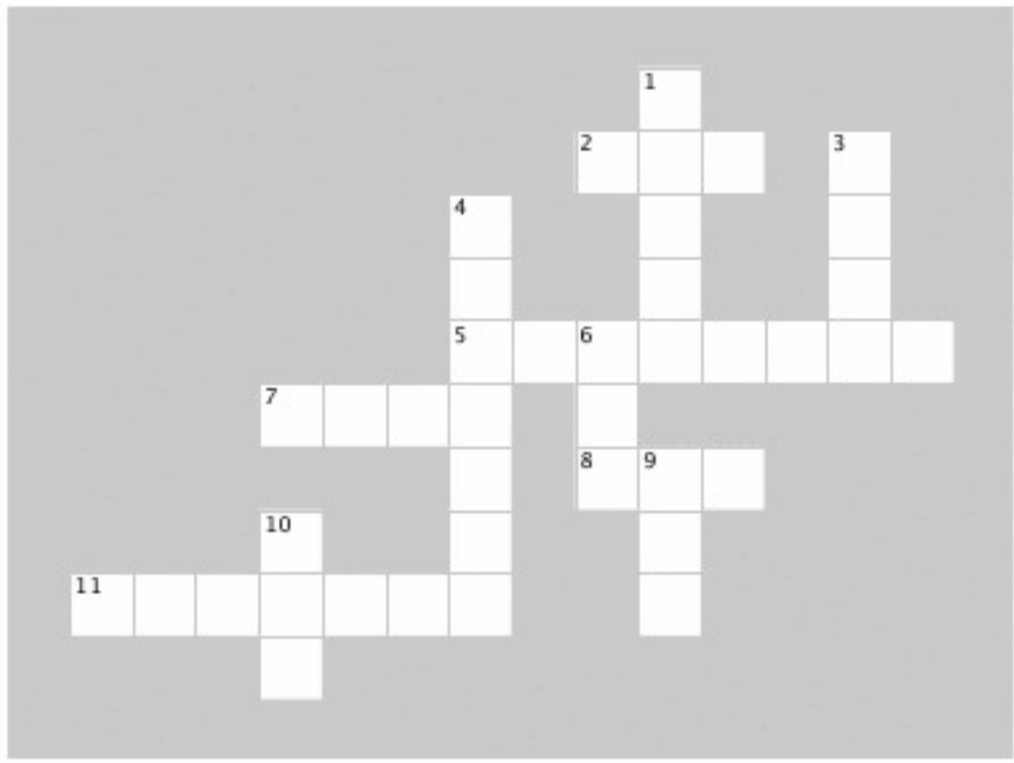
Rappelez-vous, SQL commence à compter à 0. Donc 1 est en fait 2.

Vos requêtes s'allongent parce que vos données se compliquent.

Regardons votre table de plus près, vous l'avez peut-être poussée plus loin que ses limites. Passez au chapitre 7...

SELECT-croisés

Il est temps de laisser votre cerveau droit faire une pause et de mettre votre hémisphère gauche au travail : tous les mots sont liés à SQL et viennent de ce chapitre.



Définitions horizontales :

2. Vous pouvez trouver la plus petite valeur d'une colonne avec cette fonction.

5. Cette fonction renvoie chaque valeur une seule fois, sans duplicats.

7. Le mot-clé ... dans le CASE vous permet d'indiquer à votre SGBDR que faire si vos enregistrements ne remplissent pas les conditions.

8. Vous pouvez trouver la plus grande valeur d'une colonne avec cette fonction.

11. Utilisez ces deux mots pour regrouper des lignes à partir d'une colonne commune.

Définitions verticales :

1. Vous permet de préciser exactement le nombre de lignes à renvoyer et par laquelle commencer.

3. Si vous triez une colonne avec ORDER BY en utilisant ce mot-clé, la valeur 9 de cette colonne viendra avant le 8.

4. Utilisez ces deux mots pour trier vos résultats par ordre alphabétique selon une colonne que vous indiquez.

6. Cette fonction fait le total des valeurs numériques d'une colonne.

9. Si vous triez une colonne avec ORDER BY en utilisant ce mot-clé, la valeur 8 de cette colonne viendra avant le 9.

10. Utilisez ceci dans un SELECT pour renvoyer le nombre de résultats plutôt que les résultats eux-mêmes.

Boîte à outils SQL

Le chapitre 6 est derrière vous et toutes ces fonctions, ces mots-clés et requêtes SELECT avancés sont maintenant un jeu d'enfant pour vous. Pour une liste exhaustive des outils du livre, voyez l'annexe C.

ORDER BY

Trie vos résultats dans l'ordre alphabétique à partir d'une colonne que vous indiquez.

GROUP BY

Regroupe des lignes à partir d'une colonne commune.

COUNT

Peut vous dire combien de lignes correspondent à une requête SELECT sans que vous ayez à voir les lignes. COUNT renvoie une seule valeur entière.

DISTINCT

Renvoie chaque valeur une seule fois, sans duplicats.

AVG

Renvoie la valeur moyenne d'une colonne numérique.

MAX et MIN

Renvoie la plus grande valeur d'une colonne pour MAX et la plus petite pour MIN.

SUM

Calcule le total d'une colonne de valeurs numériques.

LIMIT

Vous permet de préciser exactement le nombre de lignes à renvoyer et avec quelle ligne commencer.

Vos nouveaux outils : des fonctions SELECT avancées, des mots-clés et des requêtes !

Qui suis-je ?

de la p. 272

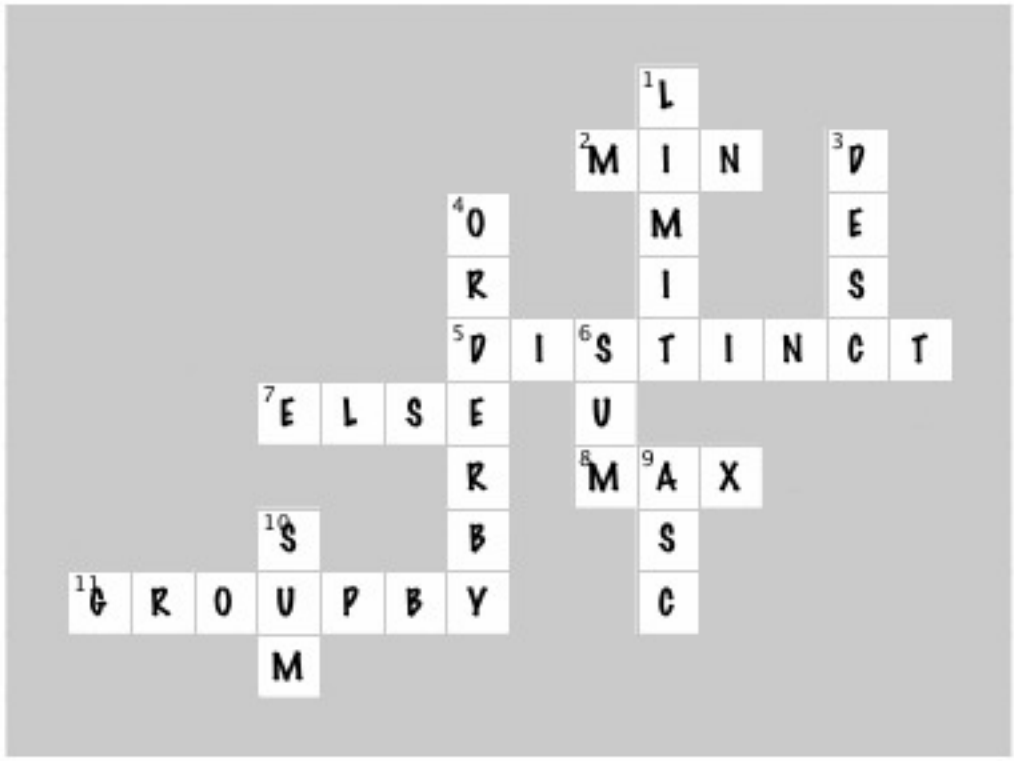
Un groupe de mots-clés et de fonctions SQL, entièrement déguisés, organisent un petit jeu, « Qui suis-je ? ». Ils vous donnent un indice et vous essayez de deviner qui ils sont à partir de ce qu'ils disent. Supposez qu'ils disent toujours la vérité. Complétez les blancs à droite pour identifier les participants. De même, pour chaque participant, notez si c'est une fonction ou un mot-clé.

Les partipants de ce soir :

COUNT, DISTINCT, AVG, MIN, GROUP BY, SUM, MAX

	Nom	fonction ou mot-clé
Le résultat que vous obtenez en m'utilisant ne vaut sans doute pas grand chose.	MIN	fonction
Ce que je recrache est plus gros que ce que je me mets dans la bouche.	SUM	fonction
Je vais vous donner des résultats absolument uniques.	DISTINCT	mot-clé
Je vais vous dire combien ils étaient.	COUNT	fonction
Vous devez m'utiliser si vous voulez obtenir une somme.	GROUP BY	mots-clés
Je ne m'intéresse qu'au plus gros nombre.	MAX	fonction
Qui suis-je ? Quelque part au milieu.	AVG	fonction

SELECT-croisés
Solution



7 conception de base de données multi-tables

Dépasser les limites de votre table

Trouver une femme pour Nico

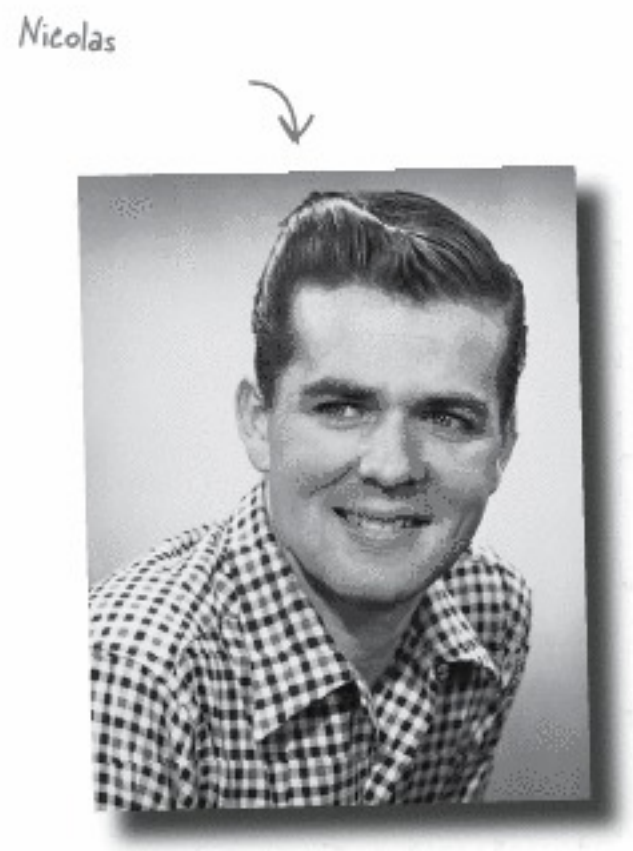
Nicolas, l'ami célibataire de Greg, lui a demandé de l'aider à trouver une femme avec des centres d'intérêt similaires. Greg commence par retrouver l'enregistrement de Nico.

Le voici :

```
id_contact : 341
nom : Bulot
prenom : Nicolas
telephone : 0411441144
email: nicobulot@vivelacampagne.com
sexe : M
anniversaire : 1975-08-28
profession : Exploitant agricole
ville : Hyères
dpmt : 83
statut : célibataire
centres_interet : animaux, équitation,
films
cherche : F célibataire
```

La colonne `centres_interet` n'est pas atomique ; elle contient plusieurs informations du même type. Il a peur que ce ne soit pas une requête facile.

Greg ajoute la requête de Nico à sa liste de tâches :



Liste de tâches

écrire une requête pour Nico : je vais écrire une requête pour interroger la colonne centres_interet. Ce sera pénible, je vais devoir utiliser **LIKE**, mais c'est seulement pour cette fois...

Pourquoi changer quoi que ce soit ?

Greg a décidé de ne rien changer à la colonne `centres_interet`. Il est disposé à écrire des requêtes compliquées parce qu'il ne pense pas devoir en écrire très souvent.

Il utilise le champ d'anniversaire `DATE` pour trouver des correspondances qui ne soient pas plus jeunes ou plus vieilles que Nico de plus de cinq ans.

À vos crayons ! Solution

Terminez la requête sur mesure de Greg pour aider Nico à trouver une femme qui convienne et qui partage tous les centres d'intérêt de Nico. Notez ce que fait chaque ligne de code.

```
SELECT * FROM mes_contacts
WHERE sexe = 'F'
AND statut = 'célibataire'
AND dpmt='83'
AND cherche LIKE '%H célibataire%'
AND anniversaire > '1970-08-28'
AND anniversaire < '1980-08-28'
AND centres_interet LIKE ..............................
AND
..............................................................
AND
..............................................................
```

À vos crayons ! Solution

Terminez la requête sur mesure de Greg pour aider Nico à trouver une femme qui convienne et qui partage tous les centres d'intérêt de Nico.
Notez ce que fait chaque ligne de code.

Sélectionne tout ce qui correspond aux conditions suivantes dans la table mes_contacts.

```
SELECT * FROM mes_contacts
WHERE sexe = 'F'
AND statut = 'célibataire'
AND dpmt ='83'
AND cherche LIKE '%H célibataire%'
AND anniversaire > '1970-08-28'
AND anniversaire < '1980-08-28'
AND centres_interet LIKE
AND
AND
```

Nico veut rencontrer une femme, nous cherchons donc une femme...

...nous voulons qu'elle soit célibataire.

...et qu'elle vive au moins dans le même départmement que Nico

Elle doit chercher un homme célibataire.

Il veut quelqu'un qui ne soit pas plus jeune ou plus vieille que lui de plus de 5 ans.

'%animaux%'

centres_interet LIKE '%équitation%'

centres_interet LIKE '%films%';

Ces lignes vont chercher des correspondances pour les centres d'intérêt de Nico. Nous aurions pu utiliser OR ici mais nous voulons des correspondances à tous ses centres d'intérêt.

La requête a très bien fonctionné

Greg a trouvé la correspondance parfaite pour Nico :

```
id_contact : 1854
nom : Fiore
prenom : Carla
telephone : 0611441144
email : cfiore@cliniqueveterinairefiore.com
sexe : F
anniversaire : 1974-01-07
profession : Vétérinaire
ville : Carqueiranne
dpmt : 83
statut : célibataire
centres_interet : équitation, films, animaux,
romans policiers, randonnée
cherche : H célibataire
```

bon âge.

bonne profession.

elle vit même pas très loin

centres d'intérêt conformes !

Carla et Déclic

Cela a trop bien fonctionné

Nico et Carla ont vraiment accroché. Greg est maintenant devenu victime de son propre succès : *tous* ses amis célibataires veulent qu'il interroge sa base de données pour eux. Et Greg a beaucoup d'amis célibataires.

Votre conception de table doit faire le travail ingrat pour vous. N'écrivez pas de requêtes tortueuses pour « contourner » une table mal conçue.

Cela prend trop de temps. Greg ajoute une note à sa liste de tâches.

Liste de tâches

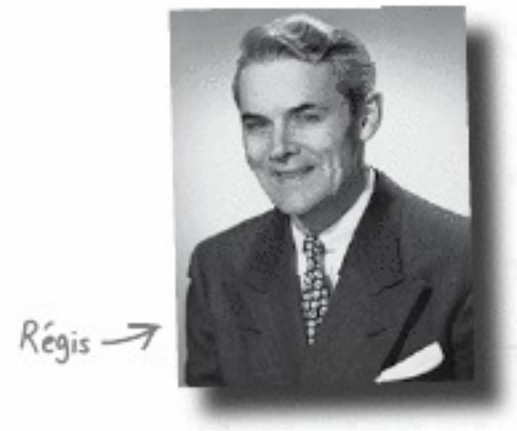
écrire une requête pour Nico : ~~je vais écrire une requête pour interroger la colonne centres_interet. Ce sera pénible, je vais devoir utiliser LIKE, mais c'est seulement pour cette fois...~~

à l'avenir, ignorer la colonne centres_interet pour avoir des requêtes plus rapides et plus faciles.

Ignorer le problème n'est pas la solution

Un autre ami, Régis, demande à Greg de lui arranger un rendez-vous. Il cherche une fille qui ne soit pas plus jeune ou plus vieille que lui de plus de cinq ans. Il vit à Tours, 37 et a d'autres centres d'intérêt que Nico.

Greg décide de ne pas se préoccuper de la colonne `centres_interet` pour que ses requêtes restent courtes et simples.



Écrivez une requête pour Régis sans utiliser la colonne centres_interet.

```
id_contact : 873
nom : Bistrot
prenom : Régis
telephone : 0212211221
email : moi@moniqueestunflan.com
sexe : H
anniversaire : 1955-03-20
profession : Comédien
ville : Tours
dpmt : 37
statut : célibataire
centres_interet : animaux, collection de pogs,
course aux trésors
cherche : F célibataire
```

→ Réponses page 342.

Trop de fausses correspondances

Greg donne une longue liste de correspondances à Régis. Après quelques semaines, Régis appelle Greg pour lui dire que sa liste est inutile et qu'aucune des femmes n'avait quoi que ce soit en commun avec lui.

Liste de tâches

écrire une requête pour Nico : je vais écrire une requête pour interroger la colonne centres_interet. Ce sera pénible, je vais devoir utiliser **LIKE**, mais c'est seulement pour cette fois...

à l'avenir, ignorer la colonne centres_interet pour avoir des requêtes plus rapides et plus faciles.

n'interroger que le premier centre d'intérêt et ignorer le reste des informations de cette colonne.

Les centres d'intérêt SONT importants. Nous ne pouvons pas les ignorer, il s'y trouve des informations précieuses.

N'utiliser que le premier centre d'intérêt

Greg sait maintenant qu'il ne peut pas ignorer tous les centres d'intérêt. Il suppose que les gens lui ont donné leurs centres d'intérêt par ordre d'importance et décide qu'il ne va chercher que le premier. Ses requêtes sont toujours assez pénibles à écrire, mais pas autant que quand il ajoutait `LIKE` pour tous les centres d'intérêt de la colonne `centres_interet`.

À vos crayons ! Solution

Utilisez la fonction SUBSTRING_INDEX pour n'obtenir que le premier centre d'intérêt de la colonne centres_interet.

À vos crayons ! Solution

Utilisez la fonction SUBSTRING_INDEX pour n'obtenir que le premier centre d'intérêt de la colonne centres_interet.

SUBSTRING_INDEX(centres_interet, ',', 1)

C'est «1» parce que l'on cherche la première virgule. Si c'était «2», la fonction continuerait jusqu'à avoir trouvé une deuxième virgule et prendrait tout ce qui se trouve devant elle, c'est à dire les deux premiers centres d'intérêt.

Ceci prend tout ce qui se trouve devant la virgule dans la colonne centres_interet, soit une sous-chaîne.

Voici la virgule que cherche la commande.

Puis Greg écrit une requête pour aider Régis à trouver un rendez-vous en utilisant son `SUBSTRING_INDEX` et en précisant que le premier centre d'intérêt doit correspondre à 'animaux'.

```
SELECT * FROM mes_contacts
WHERE sexe = 'F'
AND statut = 'célibataire'
AND dpmt='37'
AND cherche LIKE '%H célibataire%'
AND anniversaire > '1950-03-20'
AND anniversaire < '1960-03-20'
AND SUBSTRING_INDEX(centres_interet,',',1) = 'animaux';
```

Seules les femmes qui ont 'animaux' listé en premier dans leurs centres d'intérêt vont s'afficher dans les résultats.

Une correspondance possible

Enfin ! Greg a trouvé une correspondance pour Régis :

```
id_contact : 459
nom : Perrault
prenom : Alexandra
telephone : 0234455667
email : angealex@yahoo.com
sexe : F
anniversaire : 1956-09-19
profession : Artiste
ville : Courçay
dpmt : 37
statut : célibataire
centres_interet : animaux
cherche : H célibataire
```

bon âge

vit près de chez Régis

centre d'intérêt correspondant

Mal assortis

Régis a invité Alexandra à sortir et Greg attendait avec anxiété de savoir comment cela s'était passé. Il commençait à imaginer sa table `mes_contacts` comme le début d'un grand site de rencontre.

Le jour suivant, Régis sonne à la porte de Greg, visiblement énervé.

Régis crie : « C'est sûr, elle s'intéresse aux animaux. Mais tu ne m'avais pas dit qu'un de ses centres d'intérêt était la taxidermie. Des animaux morts partout ! »

Liste de tâches

~~**écrire une requête pour Nico** : je vais écrire une requête pour interroger la colonne centres_interet. Ce sera pénible, je vais devoir utiliser **LIKE**, mais c'est seulement pour cette fois...~~

~~**à l'avenir, ignorer la colonne centres_interet** pour avoir des requêtes plus rapides et plus faciles.~~

~~**n'interroger que le premier centre d'intérêt** et ignorer le reste des informations de cette colonne.~~

créer plusieurs colonnes pour avoir un centre d'intérêt dans chaque parce qu'avoir tous les centres d'intérêt dans une colonne rend les requêtes difficiles.

La correspondance parfaite pour Régis était dans la table, mais n'a jamais été découverte parce que ses centres d'intérêt se trouvaient dans un ordre différent.

Greg décide de repenser sa table.

MUSCLEZ VOS NEURONES

À quoi va ressembler la prochaine requête de Greg quand il aura ajouté plusieurs colonnes centres_interet ?

Ajouter plus de colonnes de centres d'intérêt

Greg réalise que n'avoir qu'une seule colonne centres_interet rend l'écriture de ses requêtes inexacte. Il doit utiliser `LIKE` pour essayer de faire correspondres les centres d'intérêt, obtenant parfois de fausses correspondances.

Comme il a récemment appris à utiliser `ALTER` pour modifier une table, ainsi que la façon de diviser des chaînes de texte, il décide de créer plusieurs colonnes centre_interet et de mettre un centre d'intérêt par colonne. Il pense que quatre colonnes devraient suffire.

À vos crayons !

Utilisez votre ALTER et la fonction SUBSTRING_INDEX pour obtenir ces colonnes. Écrivez autant de requêtes que nécessaire.

```
id_contact
nom
prenom
telephone
email
sexe
anniversaire
profession
ville
dpmt
statut
centre_interet1
centre_interet2
centre_interet3
centre_interet4
cherche
```

→ Réponses page 341.

Recommencer

Greg se sent coupable de ce qui est arrivé à Régis et Alexandra, il va donc faire un autre essai. Il commence par retrouver l'enregistrement de Régis :

```
id_contact : 873
nom : Bistrot
prenom : Régis
telephone : 0212211221
email : moi@moniqueestunflan.com
sexe : H
anniversaire : 1955-03-20
profession : Comédien
ville : Tours
dpmt : 37
statut : célibataire
centre_interet1 : animaux
centre_interet2 : collection de pogs
centre_interet3 : course aux trésors
centre_interet4 : NULL
cherche : F célibataire
```

Quatre colonnes centre_interet dans notre table fraîchement reformatée

Exercices

Puis Greg écrit une requête sur mesure pour aider Régis à trouver quelqu'un qui lui convienne. Il y met tout ce à quoi il peut penser pour avoir une bonne correspondance. Il commence par les colonnes les plus simples : sexe, statut, dpmt, cherche et anniversaire, avant d'interroger toutes ces colonnes centre_interet.

Écrivez cette requête ici.

Puis Greg écrit une requête sur mesure pour aider Régis à trouver quelqu'un qui lui convienne. Il y met tout ce à quoi il peut penser pour avoir une bonne correspondance. Il commcence par les colonnes les plus simples : sexe, statut, dpmt, cherche et anniversaire, avant d'interroger toutes ces colonnes centre_interet.

Écrivez cette requête ici.

```
SELECT * FROM mes_contacts

WHERE sexe = 'F'
AND statut = 'célibataire'
AND dpmt='37'
AND cherche LIKE '%H célibataire%'
AND anniversaire > '1950-03-20'
AND anniversaire < '1960-03-20'
AND
(
centre_interet1 = 'animaux'
OR centre_interet2 = 'animaux'
OR centre_interet3 = 'animaux'
OR centre_interet4 = 'animaux'
)
AND
(
centre_interet1 = 'collection de pogs'
OR centre_interet2 = 'collection de pogs'
OR centre_interet3 = 'collection de pogs'
OR centre_interet4 = 'collection de pogs'
)
AND
(
centre_interet1 = 'course aux trésors'
OR centre_interet2 = 'course aux trésors'
OR centre_interet3 = 'course aux trésors'
OR centre_interet4 = 'course aux trésors'
);
```

Régis veut sortir avec une femme célibataire née entre 1950 et 1960, qui vit en Indre et Loire et cherche un homme célibataire.

Greg doit chercher dans toutes les colonnes centre_interet pour voir si les valeurs correspondent à celles de Régis puisqu'il peut y avoir une correspondance dans chacune des nouvelles colonnes.

Régis a une valeur NULL pour centre_interet4, nous n'avons donc que trois centres_interet à vérifier au lieu de quatre.

Tout est perdu...

Ajouter les nouvelles colonnes n'a rien fait pour résoudre le problème de base ; la conception de la table ne facilite pas son interrogation. Chaque version de la table enfreint les règles des données atomiques.

Liste de tâches

écrire une requête pour Nico : je vais écrire une requête pour interroger la colonne centres_interet. Ce sera pénible, je vais devoir utiliser **LIKE**, mais c'est seulement pour cette fois...

à l'avenir, ignorer la colonne centres_interet pour avoir des requêtes plus rapides et plus faciles.

n'interroger que le premier centre d'intérêt et ignorer le reste des informations de cette colonne.

créer plusieurs colonnes pour avoir un centre d'intérêt dans chaque parce qu'avoir tous les centres d'intérêt dans une colonne rend les requêtes difficiles.

?

Cela semblait une si bonne solution. Mais cela complique encore plus les requêtes.

...Mais attendez

Peut-on créer une table ne contenant que les centres d'intérêt ? Cela peut-il nous aider ?

MUSCLEZ VOS NEURONES

Ajouter une table nous aiderait-il ? Comment pourrions-nous relier les données de la nouvelle table à notre table actuelle ?

Penser au-delà d'une seule table

Nous savons qu'il n'y a pas de bonne solution si nous travaillons sur la table actuelle. Nous avons essayé plusieurs méthodes pour arranger les données, jusqu'à altérer la structure de la table. Rien n'a fonctionné.

Nous devons penser au-delà de cette table. Ce dont nous avons réellement besoin, c'est de **plus de tables** qui peuvent ***fonctionner avec*** la table actuelle pour nous permettre d'**associer chaque personne avec plus d'un centre d'intérêt**. Et cela nous permettra de garder les données existantes intactes.

Nous devons déplacer les colonnes non-atomiques de notre table dans de nouvelles tables.

Fichier Edition Fenêtre Aide TableDésordonnée

```
> DESCRIBE mes_contacts;
+-----------------+--------------+------+-----+---------+----------------+
| Field           | Type         | Null | Key | Default | Extra          |
+-----------------+--------------+------+-----+---------+----------------+
| id_contact      | int(11)      | NO   | PRI | NULL    | auto_increment |
| nom             | varchar(30)  | YES  |     | NULL    |                |
| prenom          | varchar(20)  | YES  |     | NULL    |                |
| telephone       | varchar(10)  | YES  |     | NULL    |                |
| email           | varchar(50)  | YES  |     | NULL    |                |
| sexe            | char(1)      | YES  |     | NULL    |                |
| anniversaire    | date         | YES  |     | NULL    |                |
| profession      | varchar(50)  | YES  |     | NULL    |                |
| ville           | varchar(50)  | YES  |     | NULL    |                |
| dpmt            | varchar(2)   | YES  |     | NULL    |                |
| statut          | varchar(20)  | YES  |     | NULL    |                |
| centres_interet | varchar(100) | YES  |     | NULL    |                |
| cherche         | varchar(100) | YES  |     | NULL    |                |
+-----------------+--------------+------+-----+---------+----------------+
13 rows in set (0.01 sec) >
```

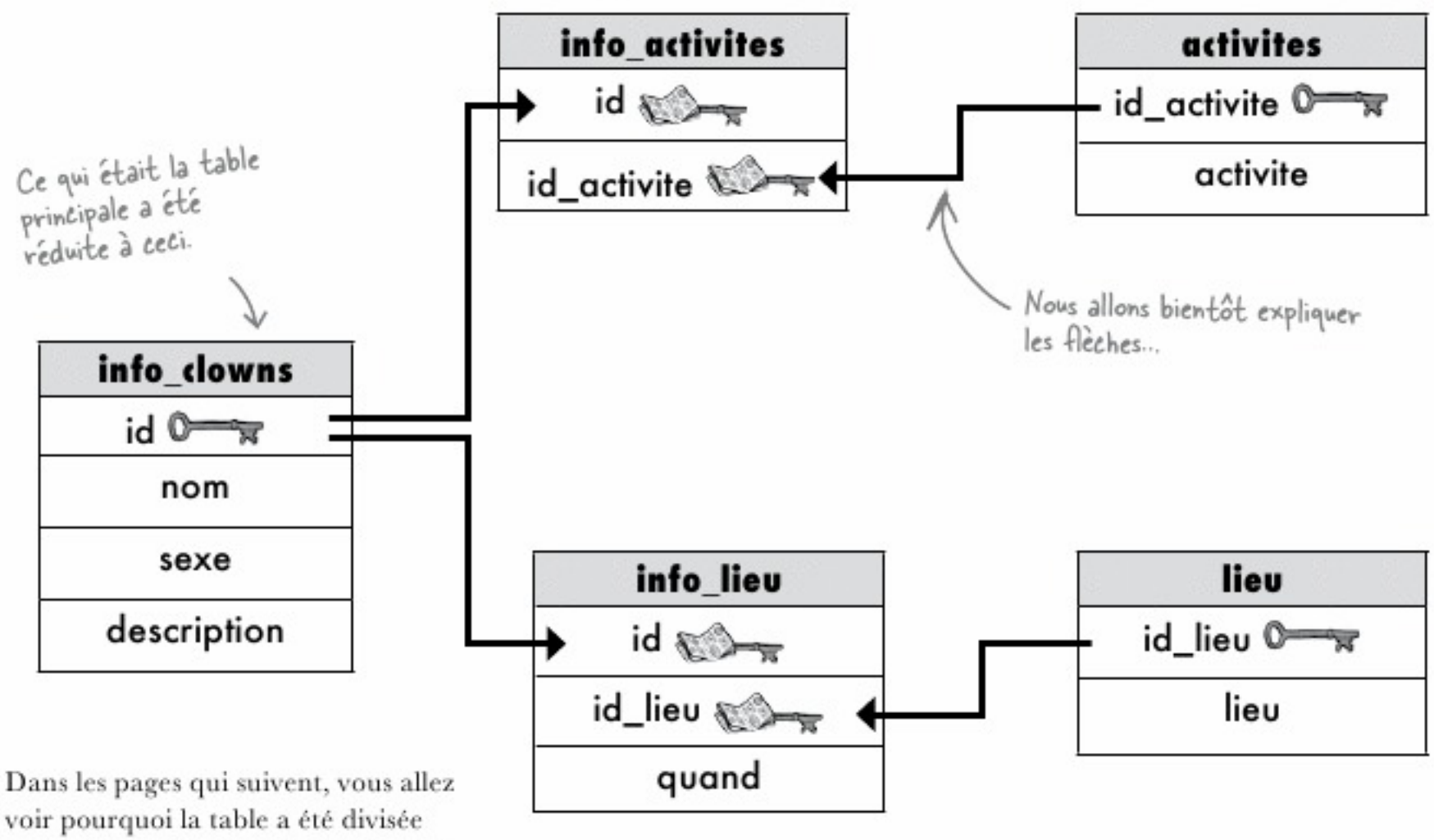
La base de données multi-tables d'espionnage de clowns

Vous souvenez-vous de notre espionnage de clowns du chapitre 3 ? Le problème des clowns de Dataville continue à se développer, nous avons donc altéré la table en un ensemble de tables bien plus utile.

Ce à quoi ressemblait l'ancienne table info_clowns.

info_clowns

nom	vu_en_dernier	description	activites
Elsie	Maison de retraite La colline des cerises	F, cheveux rouges, robe verte, pieds immenses	ballons, petite voiture
Cornichon	Fête de Max Leblanc	H, cheveux [illegible]nges, costume bleu, pieds immenses	mime
[illegible]	[illegible]	[illegible]	[illegible]



Dans les pages qui suivent, vous allez voir pourquoi la table a été divisée de cette façon et ce que signifient les flèches et les clés. Quand nous aurons terminé, nous pourrons appliquer les mêmes principes à la `liste_greg`.

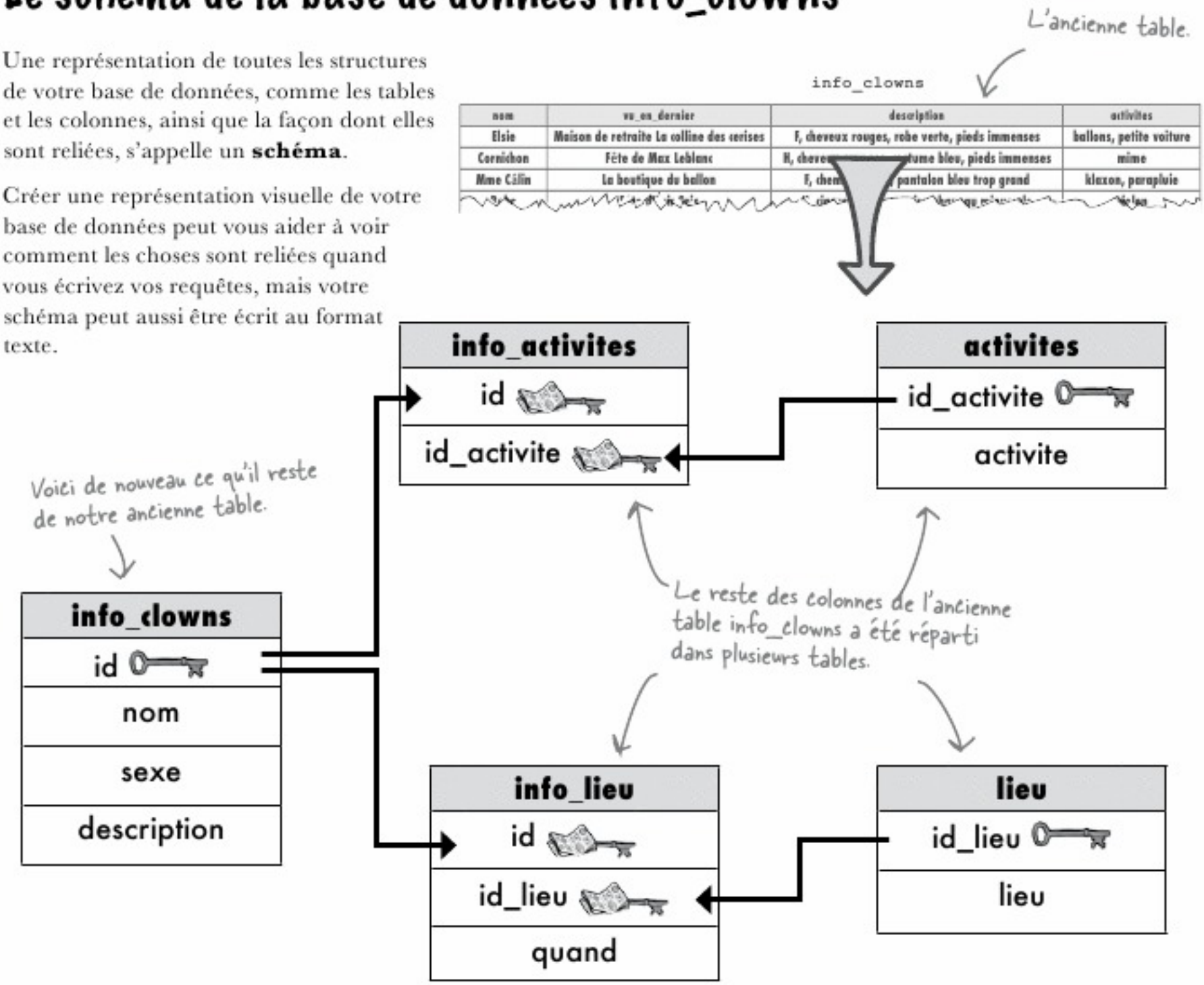
MUSCLEZ VOS NEURONES

Que signifient les fléches, à votre avis ?
Et ces symboles de clés ?

Le schéma de la base de données info_clowns

Une représentation de toutes les structures de votre base de données, comme les tables et les colonnes, ainsi que la façon dont elles sont reliées, s'appelle un **schéma**.

Créer une représentation visuelle de votre base de données peut vous aider à voir comment les choses sont reliées quand vous écrivez vos requêtes, mais votre schéma peut aussi être écrit au format texte.



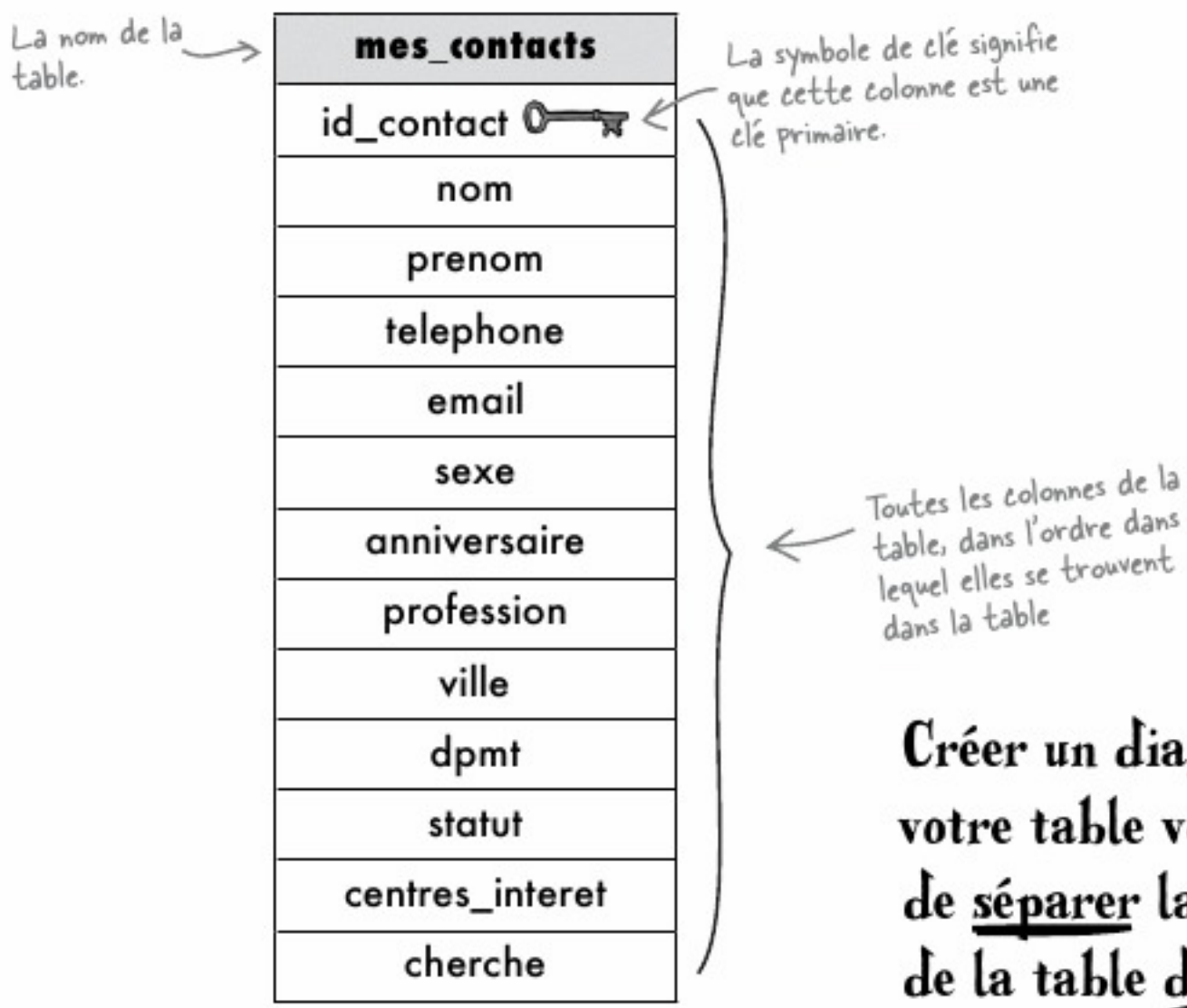
Une description des données (les colonnes et les tables) de votre base de données, ainsi que de tout autre objet en dépendant et de la façon dont ils sont tous liés entre eux s'appelle un SCHÉMA.

Un moyen plus simple de schématiser vos tables

Vous avez vu comment la table info_clowns a été convertie. Voyons comment nous pouvons arranger la table `mes_contacts` de la même façon.

Jusqu'ici, à chaque fois que nous avons regardé la table, soit nous l'avons représentée avec les noms de colonnes en haut et les données en dessous, soit nous avons utilisé une instruction `DESCRIBE` dans la fenêtre du terminal. Ces deux façons conviennent bien aux tables simples mais ne sont pas très pratiques si nous voulons créer un diagramme de plusieurs tables.

Voici une technique de sténo pour schématiser la table `mes_contacts` actuelle :



Créer un diagramme de votre table vous permet de séparer la conception de la table des données qui s'y trouvent.

Comment passer d'une table à deux

Nous savons que la colonne centres_interet est très difficile à interroger telle qu'elle est pour l'instant. Elle a plusieurs valeurs dans la même colonne et même quand nous avons essayé d'en faire plusieurs colonnes, nos requêtes étaient assez difficiles à écrire.

Voici notre table `mes_contacts` actuelle. Notre colonne centres_interet n'est pas atomique et il n'y a vraiment qu'un moyen de la rendre atomique : nous avons besoin d'une nouvelle table pour tous les centres d'intérêt.

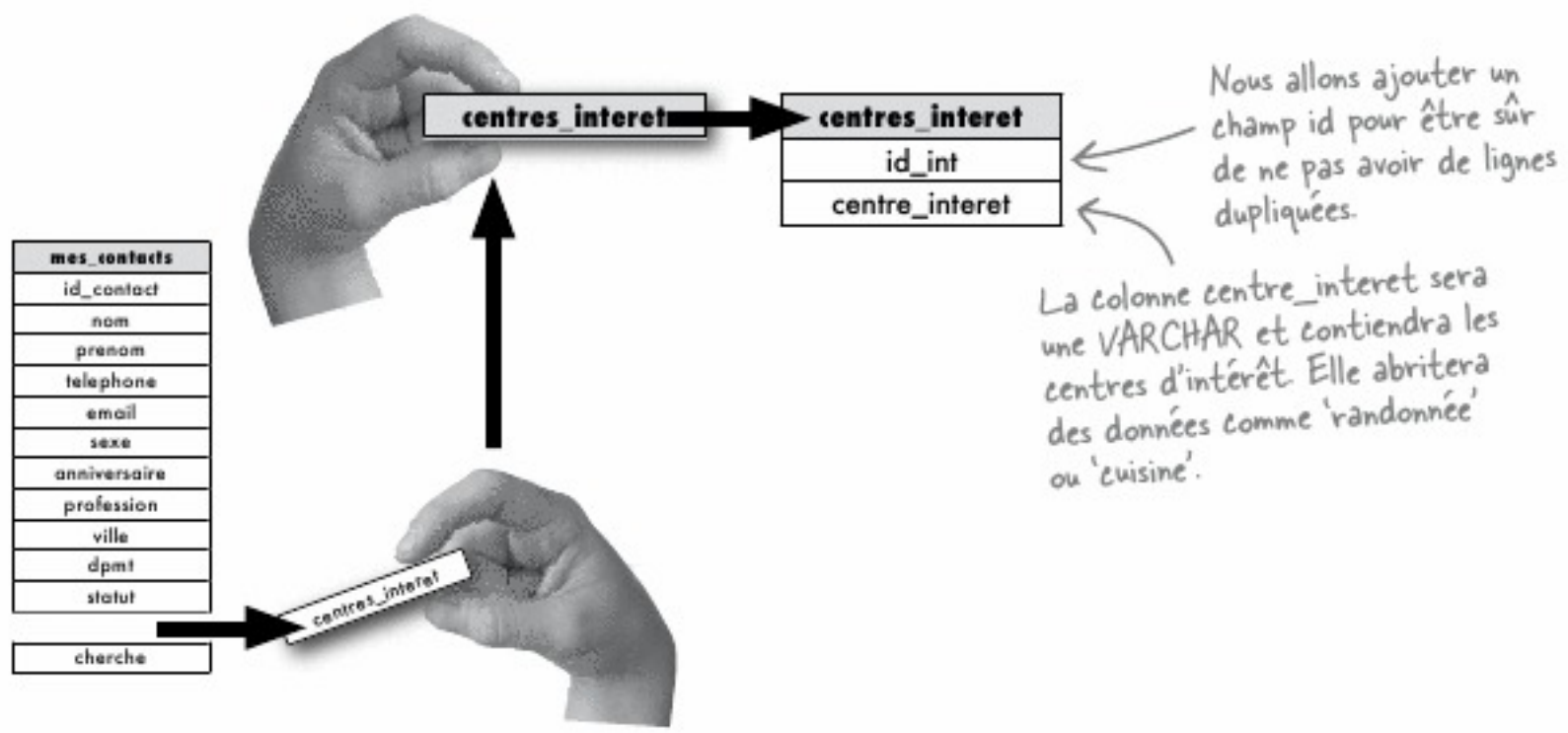
Nous allons commencer par dessiner quelques diagrammes de ce à quoi pourraient ressembler nos tables. Nous n'allons pas réellement créer notre nouvelle table ni toucher à aucune donnée jusqu'à ce que nous trouvions notre nouveau schéma.

mes_contacts
id_contact
nom
prenom
telephone
email
sexe
anniversaire
profession
ville
dpmt
statut
centres_interet
cherche

Voici mes_contacts. Elle n'est pas encore atomique.

1 **Enlever la colonne centres_interet et la placer dans sa propre table.**

Ici nous avons déplacé la colonne centres_interet dans une nouvelle table.

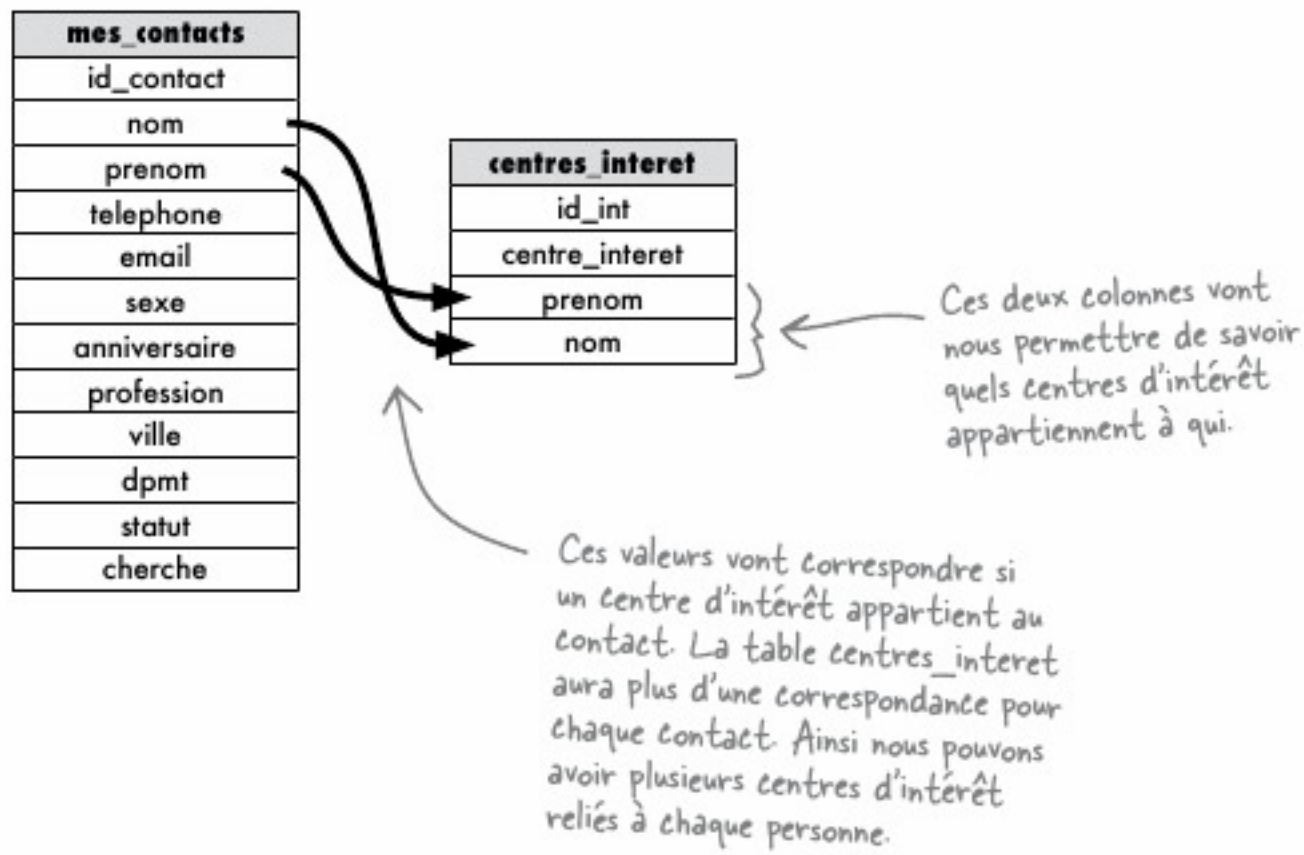


Notre nouvelle table centres_interet va abriter tous les centres d'intérêt de la table `mes_contacts`, un centre d'intérêt par ligne.

2 Ajouter des colonnes qui nous permettront d'identifier quels centre d'intérêt appartiennent à quelle personne de la table mes_contacts.

Nous avons enlevé notre centres_interet de `mes_contacts`, mais nous n'avons aucun moyen de savoir quels centres d'intérêt appartiennent à quelle personne. Nous devons utiliser les informations de la table `mes_contacts` et les placer dans la table centres_interet pour relier ces table entre elles.

Une possibilité est d'ajouter les colonnes `prenom` et `nom` à la table centres_interet.



MUSCLEZ VOS NEURONES

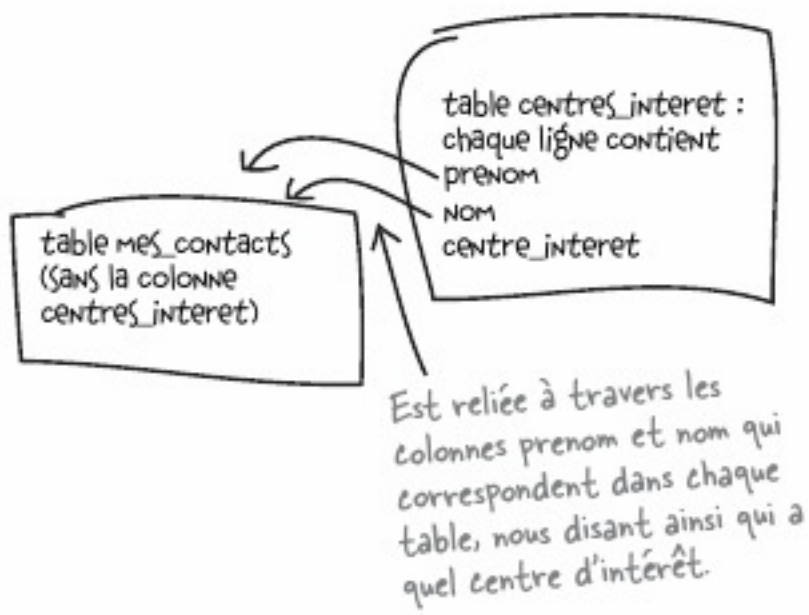
L'idée est bonne, mais prenom et nom ne sont pas les meilleurs choix de colonnes pour relier ces tables.

Pourquoi ?

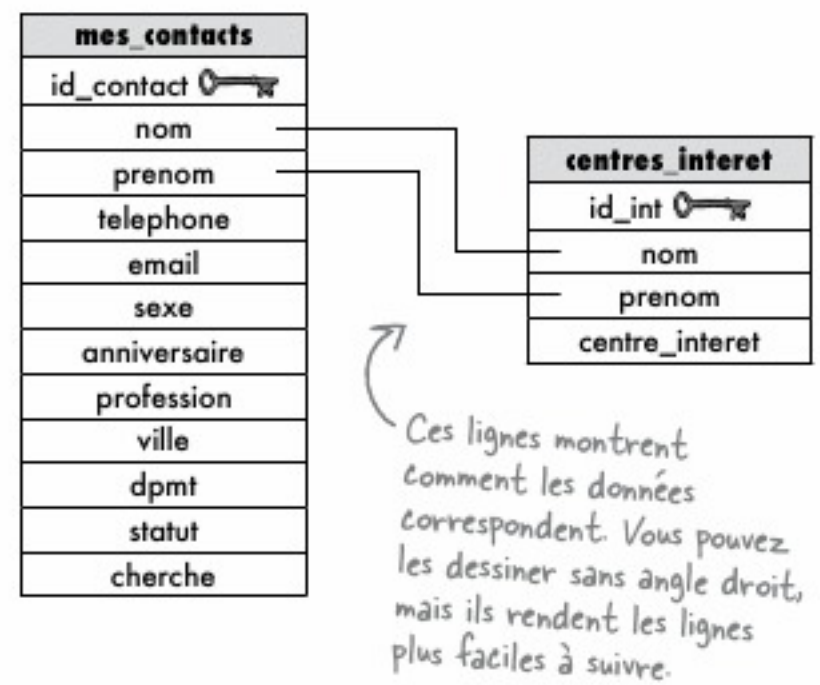
Relier vos tables dans un diagramme

Regardons notre idée pour la table `mes_contacts` de plus près.

Voici notre brouillon de départ :



Et voici notre nouveau schéma :



Remarquez que les lignes avec angle droit reliant les tables montrent les colonnes qui coïncident dans chaque table Le schéma nous permet d'ordonner notre croquis d'une façon que tout développeur SQL peut comprendre parce qu'il utilise des symboles standards.

Et voici une série d'instructions `SELECT` qui vont nous permettre d'utiliser les données des deux tables.

❶
```
SELECT prenom, nom
FROM mes_contacts
WHERE (des conditions);
```

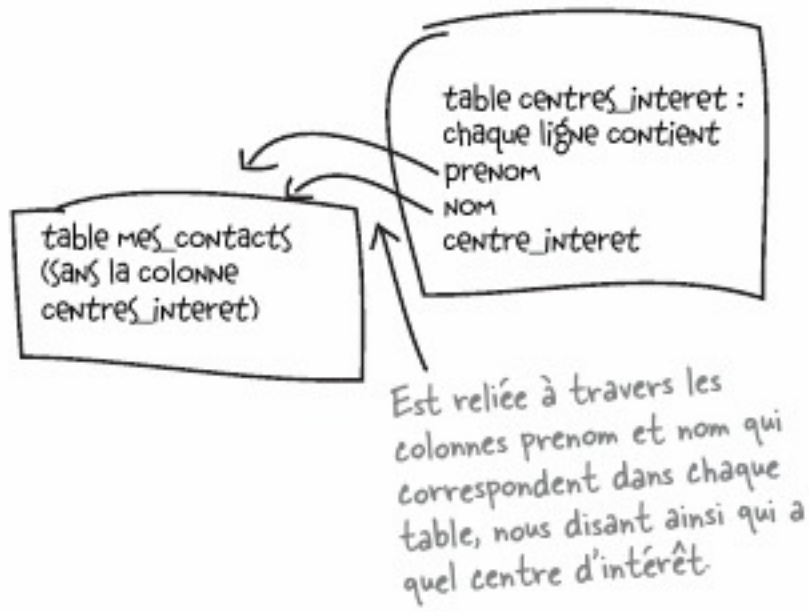
❷
```
SELECT centre_interet
FROM centres_interet
WHERE prenom = 'Unprenom'
AND nom = 'Unnom';
```

Ne vous inquiétez pas si cela semble inefficace. C'est seulement pour vous montrer comment les données d'une table peuvent être utilisées pour récupérer des données d'une autre. (Nous allons bientôt vous montrer une meilleure méthode.)

À vos crayons !

Utilisez cet espace pour schématiser d'autres idées pour ajouter de nouvelles tables à la base de données liste_greg et nous aider à conserver plusieurs centres d'intérêt.

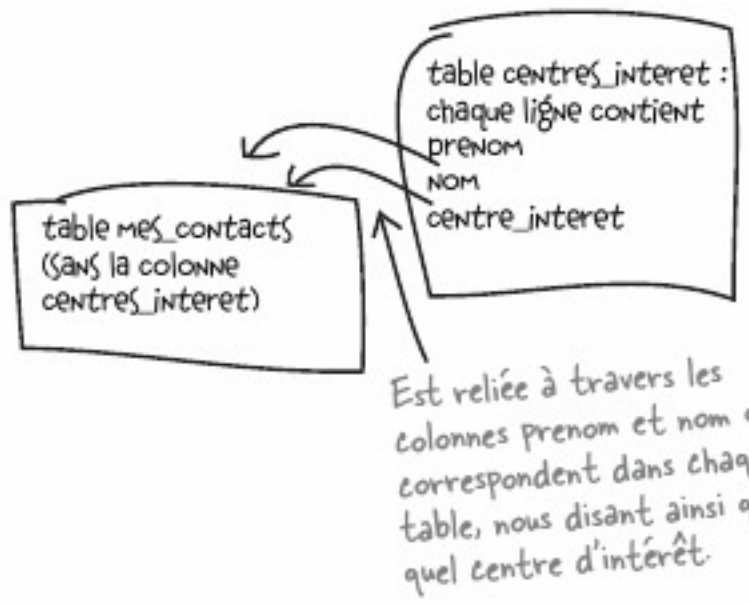
Vous n'avez pas besoin d'un dessin aussi soigné que notre schéma ; nous en sommes à l'étape des idées. Une idée est déjà dessinée pour vous, mais elle a un défaut.



À vos crayons ! Solution

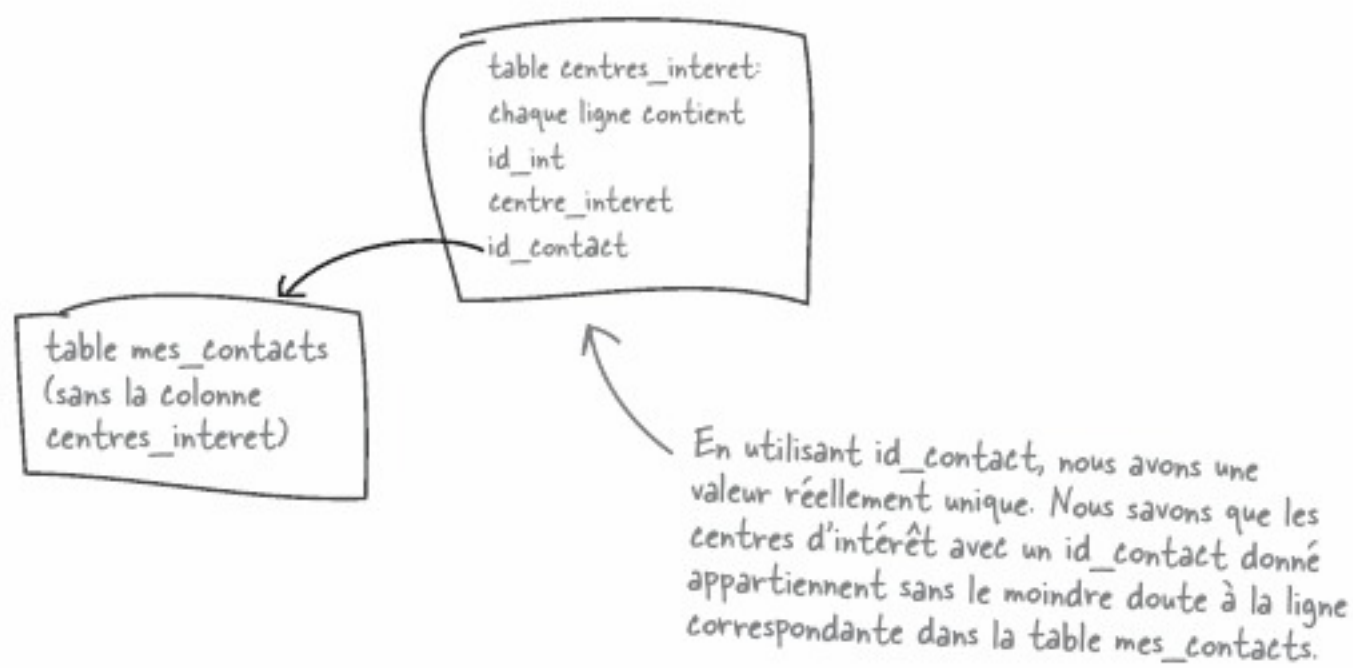
Utilisez cet espace pour schématiser d'autres idées pour ajouter de nouvelles tables à la base de données liste_greg et nous aider à conserver plusieurs centres d'intérêt.

Vous n'avez pas besoin d'un dessin aussi soigné que notre schéma ; nous en sommes à l'étape des idées. Une idée est déjà dessinée pour vous, mais elle a un défaut.



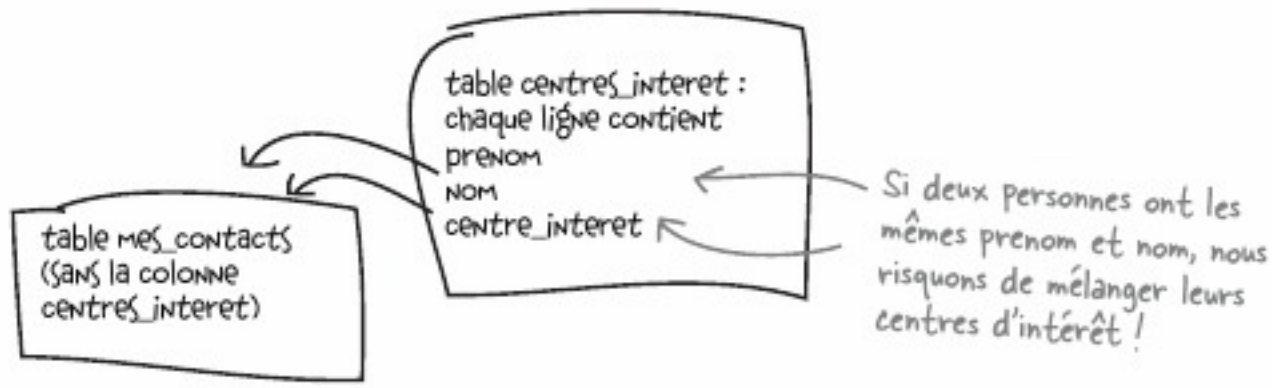
Utiliser le prénom et le nom pour se lier à la table centres_interet n'est pas une très bonne idée. Plus d'une personne dans mes_contacts peuvent partager les mêmes nom et prénom, il est donc possible que l'on relie des personnes aux mauvais centres d'intérêt. Nous ferions mieux d'utiliser notre clé primaire pour faire le lien.

Plutôt que d'utiliser les prénom et nom qui peuvent ne pas être véritablement uniques, nous pouvons utiliser id_contact pour relier nos tables :



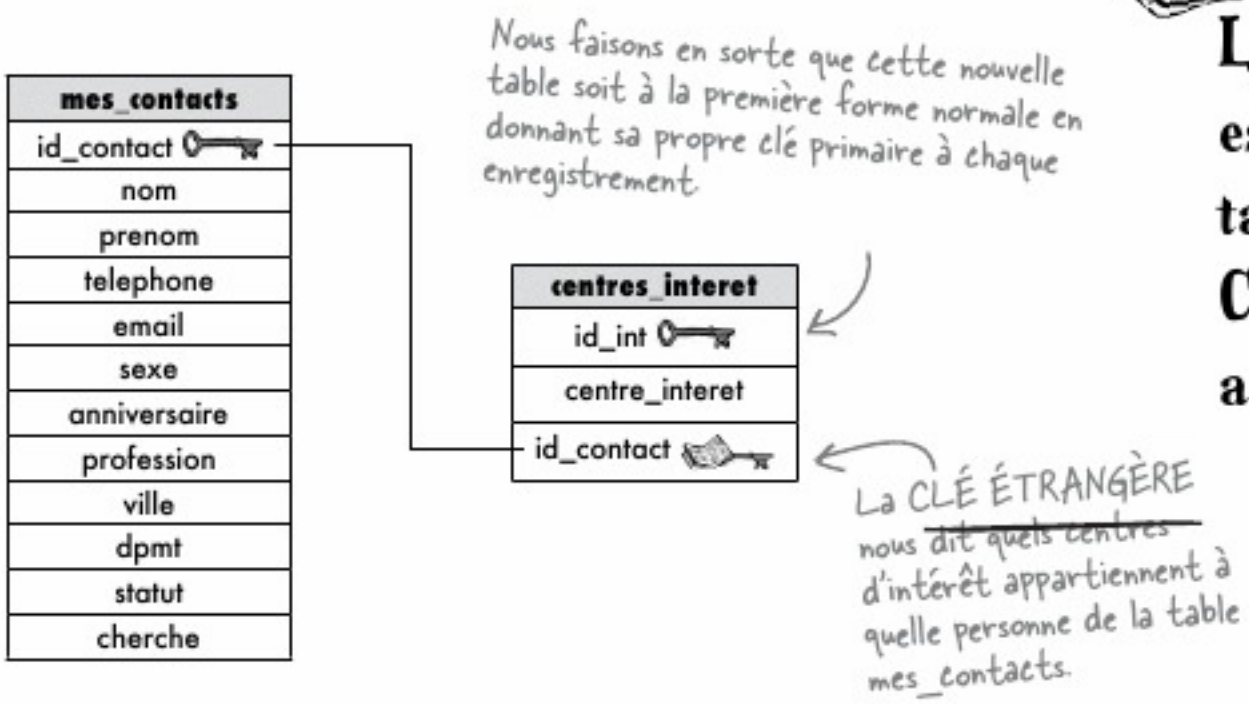
Relier vos tables

Notre premier schéma de tables reliées pose un problème : nous essayons d'utiliser les champs `prenom` et `nom` pour relier les tables. Mais qu'arrive-t-il si deux personnes de la table `mes_contacts` ont les mêmes `prenom` et `nom` ?



Il nous faut une colonne **unique** pour les relier. Heureusement, comme nous avons déjà commencé à normaliser `mes_contacts`, nous avons une colonne véritablement unique : la **clé primaire**.

Nous pouvons utiliser la valeur de la clé primaire de la table `mes_contacts` en tant que colonne dans la table `centres_interet`. Mieux encore, nous saurons quels centres d'intérêt appartiennent à quelle personne de la table `mes_contacts` grâce à cette colonne. Cela s'appelle une **clé étrangère**.



La CLÉ ÉTRANGÈRE est une colonne de la table qui référence la CLÉ PRIMAIRE d'une autre table.

À savoir sur les clés étrangères

Une clé étrangère peut avoir un nom différent de la clé primaire dont elle provient.

La clé primaire utilisée par une clé étrangère s'appelle également une *clé parente*. La table d'où vient la clé primaire est appelée la table parente.

La clé étrangère peut s'utiliser pour vérifier que les lignes d'une table ont des lignes correspondantes dans une autre table.

Les valeurs d'une clé étrangère peuvent être null, même si celles d'une clé primaire ne le peuvent pas.

Les clés étrangères ne doivent pas être uniques ; en réalité, elles ne le sont souvent pas.

Une clé étrangère NULL signifie qu'il n'y a pas de clé primaire correspondante dans la table parente.

Mais nous pouvons faire en sorte qu'une clé étrangère contienne une valeur significative, qui existe dans la table parente, en utilisant une **contrainte**.

Contraindre votre clé étrangère

Bien que vous puissiez simplement créer une table et y placer une colonne pour jouer le rôle d'une clé étrangère, ce n'est pas réellement une clé étrangère à moins que vous ne la désigniez comme telle quand vous utilisez CREATE ou ALTER sur une table. La clé est créée à l'intérieur d'une structure appelée une **contrainte**.

Pensez à une CONTRAINTE comme à une règle que doit suivre notre table.

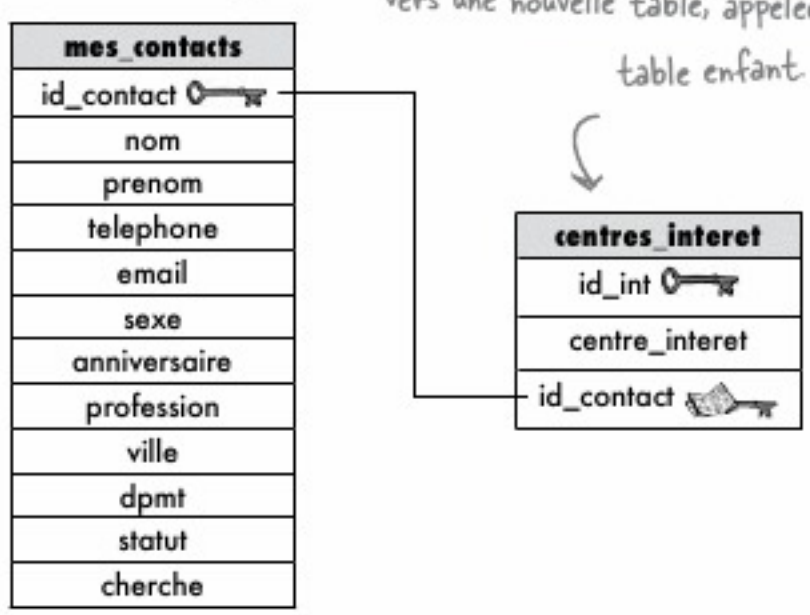
Créer une CLÉ ÉTRANGÈRE en tant que contrainte de votre table vous donne des avantages certains.

Vous aurez des erreurs si vous enfreignez les règles, ce qui vous empêchera de casser la table accidentellement.

Vous ne pouvez insérer dans votre clé étrangère que des valeurs qui existent dans la table d'où vient la clé, la table parente. On appelle cela l'*intégrité référentielle*.

Notre table mes_contacts d'origine est maintenant une table parente puisqu'une partie de ses données a été transférée vers une nouvelle table, appelée une....

table enfant



L'intégrité référentielle signifie que vous ne pouvez placer dans la clé étrangère de la table enfant que des valeurs qui existent déjà dans la table parente.

Vous pouvez utiliser une clé étrangère pour référencer une valeur unique de la table parente.

Ce n'est pas obligatoirement la clé primaire de la table parente, mais elle doit être unique.

Comment créer des clés étrangères ?

D'accord, alors je sais qu'enlever les centres d'intérêt de mes_contacts est le seul moyen pour pouvoir les interroger plus facilement. Et Régis a vraiment besoin de rencontrer quelqu'un de sympa... Maintenant, j'ai besoin de savoir *COMMENT* créer une table avec une clé étrangère.

Vous pouvez ajouter votre clé étrangère quand vous créez votre nouvelle table.

Et vous pouvez ajouter des clés étrangères avec `ALTER TABLE`. La syntaxe est simple. Vous devez connaître le nom de la clé primaire dans la table parente ainsi que le nom de la table parente. Créons la table `centres_interet` avec une clé étrangère, `id_contact` de la table `mes_contacts`.

il n'y a pas de
Questions stupides

Q: Une fois que nous aurons enlevé les centres d'intérêt de mes_contacts, comment vais-je les interroger ?

R: Nous allons y venir dans le prochain chapitre. Et vous verrez qu'il est vraiment facile d'écrire des requêtes pour récupérer vos données de plusieurs tables. Mais pour l'instant, nous devons repenser mes_contacts pour simplifier nos requêtes et les rendre plus efficaces.

Créer une table avec FOREIGN KEY

Maintenant que vous avez vu le principe de création d'une clé étrangère avec une contrainte, voici comment vous pouvez le faire concrètement. Remarquez que nous nommons la CONSTRAINT (la contrainte) de façon à pouvoir dire de quelle table vient la clé.

Ajouter la commande PRIMARY KEY à la ligne où vous la paramétrez est un autre moyen (plus rapide) de désigner votre clé primaire.

Nous créons la clé étrangère de la même façon que n'importe quelle colonne d'index : nous la définissons comme INT et NOT NULL.

```
CREATE TABLE centres_interet (
id_int INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
centre_interet VARCHAR(50) NOT NULL,
id_contact INT NOT NULL,
CONSTRAINT mes_contacts_id_contact_ce
FOREIGN KEY (id_contact)
REFERENCES mes_contacts (id_contact)
);
```

Nous nommons cette CONSTRAINT d'une façon qui nous dise de quelle table vient la clé (mes_contacts), comment nous avons appelé la clé (id_contact) et que c'est une clé étrangère (ce).

Si nous changeons d'avis plus tard, nous utiliserons ce nom pour défaire la clé. Cette ligne est optionnelle, mais c'est une bonne pratique de l'utiliser.

Ceci précise d'où vient la clé étrangère...

et comment elle s'appelle dans l'autre table.

Le nom de colonne entre parenthèses va devenir une clé étrangère. Vous pouvez l'appeler comme vous voulez.

Ouvrez la fenêtre de votre console et saisissez le code ci-dessus pour créer votre propre table centres_interet.

Une fois que vous l'avez créée, regardez la structure de votre nouvelle table. Quelles nouvelles informations vous montrent la présence de votre contrainte ?

Ouvrez la fenêtre de votre console et saisissez le code ci-dessus pour créer votre propre table centres_interet.

Une fois que vous l'avez créée, regardez la structure de votre nouvelle table. Quelles nouvelles informations vous montrent la présence de votre contrainte ?

```
Fichier Edition Fenêtre Aide
> DESC centres_interet;
+----------------+-------------+------+-----+---------+----------------+
| Field          | Type        | Null | Key | Default | Extra          |
+----------------+-------------+------+-----+---------+----------------+
| id_int         | int(11)     | NO   | PRI | NULL    | auto_increment |
| centre_interet | varchar(50) | NO   |     |         |                |
| id_contact     | int(11)     | NO   | MUL |         |                |
+----------------+-------------+------+-----+---------+----------------+
```

MUL signifie que de multiples occurrences de la même valeur peuvent être conservées dans cette colonne. C'est ce qui nous permet de garder la trace des multiples centres_interet pour chaque id_contact de mes_contacts.

Questions stupides

il n'y a pas de

Q: Vous vous donnez beaucoup de mal pour créer une contrainte de clé étrangère, mais pourquoi ? Est-ce que vous ne pourriez pas tout simplement utiliser la clé d'une autre table et l'appeler clé étrangère sans ajouter de contrainte ?

R: On pourrait, mais en la créant comme une contrainte, vous ne pourrez y insérer que des valeurs qui existent dans la table parente. Cela impose un lien entre les deux tables.

Q: « Impose un lien »? Qu'est-ce que cela signifie ?

R: La contrainte de la clé étrangère assure l'intégrité référentielle (en d'autres termes, elle s'assure que si vous avez une ligne dans une table avec une clé étrangère, elle correspond à une ligne dans une autre table à travers la clé étrangère). Si vous essayez de supprimer la ligne dans la table de la clé primaire ou de changer une valeur de la clé primaire, vous aurez une erreur si la valeur de la clé primaire est une contrainte de clé étrangère dans une autre table.

Q: Cela veut donc dire que je ne peux jamais supprimer une ligne de mes_contacts qui a une clé primaire si elle se trouve dans la table centres_interet comme clé étrangère ?

R: Vous pouvez le faire, vous devez seulement enlever la ligne de la clé étrangère en premier. Après tout, si vous enlevez la ligne de mes_contacts, vous n'avez plus besoin de connaître les centres d'intérêt de cette personne.

Q: Mais quelle importance si j'ai ces lignes qui restent dans la table centres_interet ?

R: C'est lent. Ces lignes sont appelées orphelines et elles peuvent réellement devenir des boulets avec le temps. Elles ne font que ralentir vos requêtes parce que des informations inutiles doivent être interrogées.

Q: D'accord, je suis convaincu. Existe-t-il d'autres contraintes à part celle de la clé étrangère ?

R: Vous avez déjà vu la contrainte de la clé primaire. Utiliser le mot-clé `UNIQUE` (quand vous créez une colonne) est considéré comme une contrainte. Il existe aussi un type de contrainte, non disponible en MySQL, appelé une contrainte `CHECK`. Elle vous permet de préciser une condition qui doit être remplie par une colonne avant que vous puissiez insérer une valeur dans cette colonne. Consultez la documentation de votre SGBDR SQL pour plus d'infos sur `CHECK`.

Relations entre tables

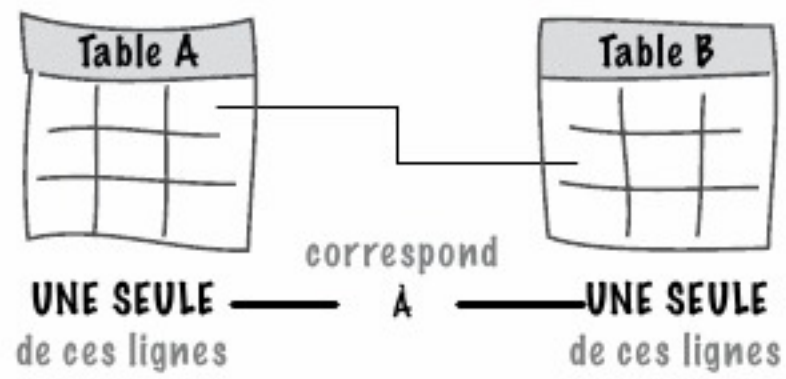
Nous savons maintenant comment relier les tables par le biais des clés étrangères, mais nous avons encore besoin d'examiner la façon dont elle sont liées les unes aux autres. Dans la table `mes_contacts`, notre problème est que nous devons associer **beaucoup de personnes** avec **beaucoup de centres d'intérêt**.

C'est l'un des trois modèles de conception (les patterns) possibles que vous retrouverez constamment avec vos données : **un-à-un**, **un-à-plusieurs** et **plusieurs-à-plusieurs**, et une fois que vous identifiez le pattern auquel correspondent vos données, trouver l'agencement de plusieurs tables, votre **schéma**, devient simple.

Patterns de données : un-à-un

Regardons le premier pattern, **un-à-un** pour voir comment il s'applique. Dans ce pattern un enregistrement de la Table A peut avoir au plus UN enregistrement correspondant dans la Table B.

Donc, disons que la Table A contienne votre nom et que la Table B contienne les détails de votre salaire et de votre numéro de sécurité sociale pour les ***isoler*** de reste de la table et les protéger davantage.



Les deux tables vont contenir votre numéro ID pour que vous obteniez la bonne fiche de paye. L'`id_employe` de la table parente est une clé primaire, l'`id_employe` de la table enfant est une clé étrangère.

Dans le schéma, la ligne de liaison est ***continue*** pour montrer que nous relions **une** chose **à une** chose.

Chaque personne de employes ne peut avoir qu'un numéro de sécurité sociale et chaque NSS ne mène qu'à une personne. Une personne, un NSS, nous avons donc une relation un-à-un.

employes

id_employe	prenom	nom
1	Patricia	Kaas
2	Mylène	Farmer
3	Carla	Bruni

salaire

nss	niveau_salaire	id_employe
234567891	2	6
345678912	5	35
123456789	7	1

Ces tables ont aussi une relation un-à-un, puisque la clé primaire de la table employes, id_employe, est utilisée comme clé étrangère de la table salaire.

Patterns de données : quand utiliser des tables un-à-un

En fait, non. Nous n'allons pas utiliser les tables un-à-un très souvent.

Il y a peu de raisons pour lesquelles vous pourriez relier vos tables par une relation un-à-un.

Quand utiliser des tables un-à-un

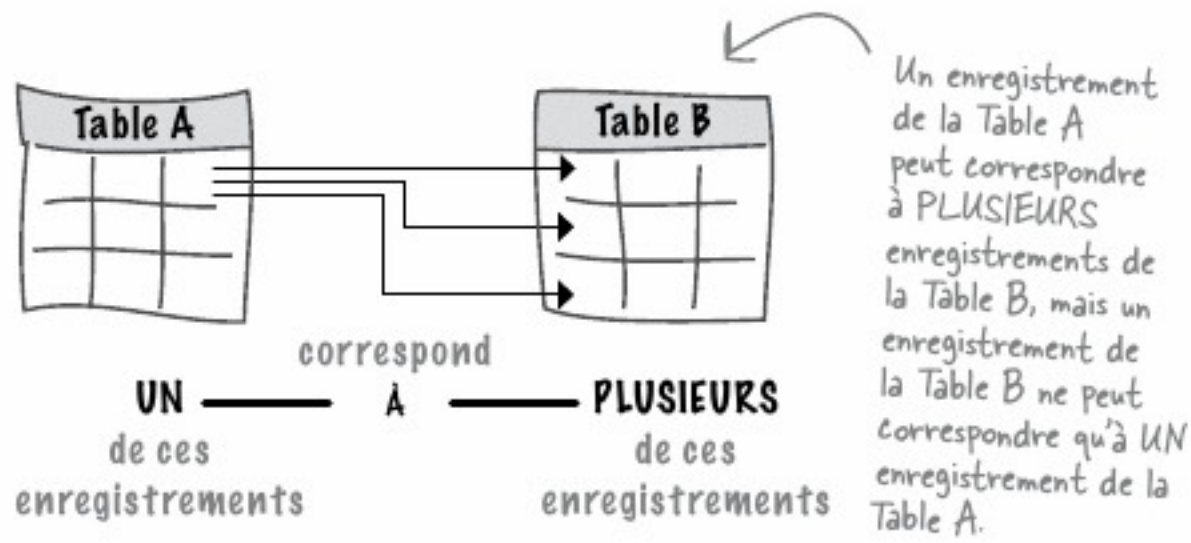
Il est généralement plus sensé de laisser vos données un-à-un dans votre table principale, mais vous pouvez parfois trouver quelques avantages à extraire ces colonnes :

1. Extraire les données peut vous permettre d'écrire des requêtes plus rapides. Par exemple, si la plupart du temps vous devez interroger le NSS et pas grand chose d'autre, vous pouvez n'interroger que la table la plus petite.
2. Si vous avez une colonne contenant des valeurs que vous ne connaissez pas encore, vous pouvez l'isoler et éviter des valeurs `NULL` dans votre table principale.
3. Si vous voulez que certaines de vos données soient moins accessibles, les isoler permet de restreindre leur accès. Par exemple, si vous avez une table d'employés, vous pouvez souhaiter conserver les informations sur leur salaire hors de la table principale.
4. Si vous avez de grosses données, un type BLOB par exemple, vous pouvez vouloir les mettre dans une table séparée.

Un-à-Un : une ligne de la table parente est reliée à une ligne de la table enfant.

Patterns de données : un-à-plusieurs

Un-à-plusieurs signifie qu'un enregistrement de la Table A peut avoir **plusieurs** enregistrements correspondants dans la Table B, mais un enregistrement de la Table B ne peut correspondre qu'à **un** enregistrement de la Table A.



Un-à-plusieurs : un enregistrement de la Table A peut avoir PLUSIEURS enregistrements correspondants dans la Table B, mais un enregistrement de la Table B ne peut correspondre qu'à UN enregistrement de la Table A.

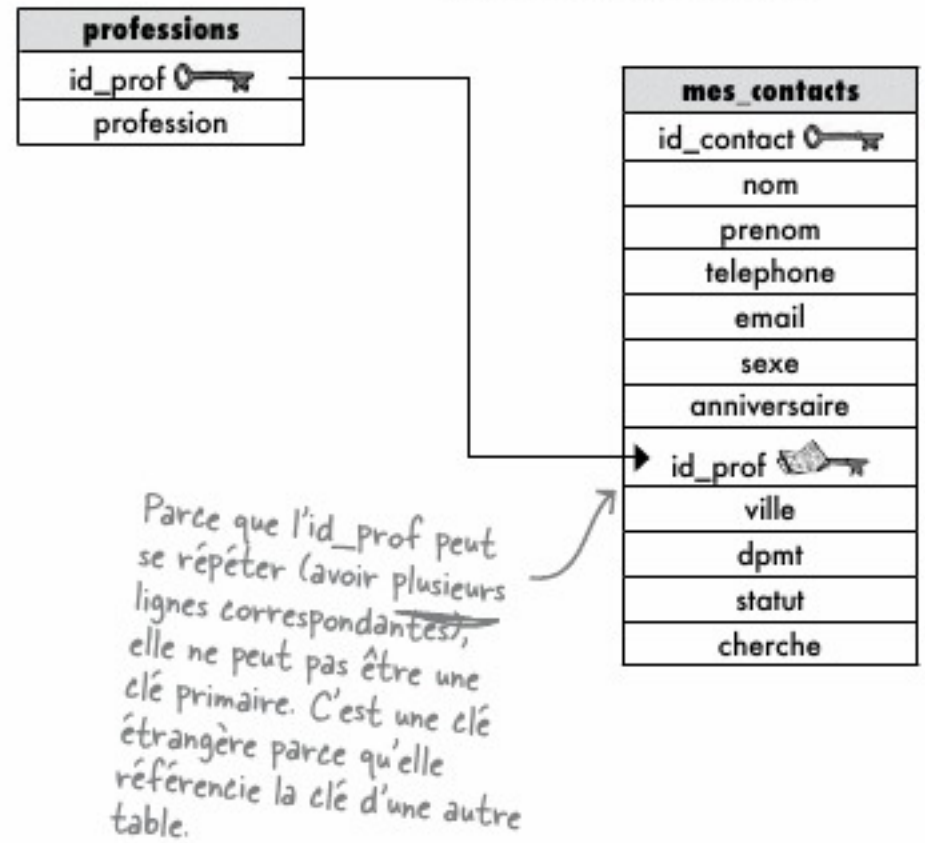
La colonne `id_prof` de `mes_contacts` est un bon exemple d'une relation un-à-plusieurs. Chaque personne n'a qu'une `id_prof`, mais plus d'une personne de `mes_contacts` peut avoir la même `id_prof`.

Dans cet exemple, nous avons déplacé la colonne `profession` dans une nouvelle table enfant et changé la colonne `profession` de la table parente en une clé étrangère, la colonne `id_prof`. Puisque c'est une relation un-à-plusieurs, nous pouvons utiliser l'`id_prof` des deux tables pour les relier.

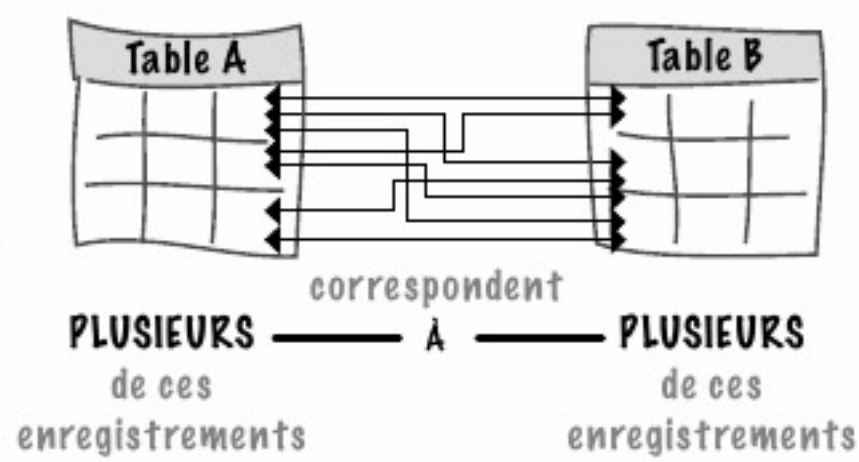
La ligne les reliant a une ***flèche noire*** à son extrémité pour montrer que nous relions **une** chose **à plusieurs** choses.

Chaque ligne de la table `professions` peut avoir plusieurs lignes correspondantes dans `mes_contacts`, mais chaque ligne de `mes_contacts` n'a qu'une ligne correspondante dans la table `professions`.

Par exemple, l'`id_prof` pour Programmeur peut apparaître plus d'une fois dans `mes_contacts`, mais chaque personne de `mes_contacts` n'aura qu'une seule `id_prof`.



Patterns de données : on arrive à plusieurs-à-plusieurs



Plusieurs femmes possèdent **plusieurs** paires de chaussures. Si nous créons une table contenant des femmes et une autre table contenant des chaussures pour garder la trace de tout cela, nous avons besoin de relier plusieurs enregistrements à plusieurs enregistrements puisque plus d'une femme peut posséder un certain type de chaussures.

Supposez que Carrie et Miranda achètent toutes les deux des tongs Old Navy et des bottes Prada, que Samantha et Miranda aient toutes les deux les sandales Manolo, et que Charlotte ait une paire de chaque et des mules en plus. Voici à quoi ressembleraient les liens entre les tables `femmes` et `chaussures`.

id_femme	femme
1	Carrie
2	Samantha
3	Charlotte
4	Miranda

id_chaussures	nom_chaussures
1	sandales Manolo
2	mules
3	tongs Old Navy
4	bottes Prada

Imaginez que ces femmes aiment tellement les chaussures qu'elles achètent toutes une paire qu'elles n'avaient pas encore. Voici à quoi ressembleraient alors les liens entre les femmes et les noms de chaussures.

Les lignes de liaison ont des flèches noires des deux côtés : nous relions plusieurs choses à plusieurs choses.

id_femme	femme
1	Carrie
2	Samantha
3	Charlotte
4	Miranda

id_chaussures	nom_chaussures
1	sandales Manolo
2	mules
3	tongs Old Navy
4	bottes Prada

MUSCLEZ VOS NEURONES

Comment pouvons-nous arranger les tables sans placer plus d'une valeur par colonne (et finir comme Greg avec ses problèmes de colonne centres_interet dans ses requêtes pour Régis) ?

À vos crayons !

Regardez cette première paire de tables. Nous avons essayé de résoudre le problème en ajoutant id_chaussures comme clé étrangère de la table des enregistrements des femmes.

id_femme	femme	id_chaussures
1	Carrie	3
2	Samantha	1
3	Charlotte	1
4	Miranda	1
5	Carrie	4
6	Charlotte	2
7	Charlotte	3
8	Charlotte	4
9	Miranda	3
10	Miranda	4

id_chaussures	nom_chaussures
1	sandales Manolo
2	mules
3	tongs Old Navy
4	bottes Prada

Les deux tables sont maintenant reliées par la colonne id_chaussures.

Dessinez vous-mêmes les tables, mais cette fois mettez l'id_femme comme clé étrangère de la table des chaussures.

Une fois terminé, dessinez les liens.

À vos crayons ! Solution

Regardez cette première paire de tables. Nous avons essayé de résoudre le problème en ajoutant id_chaussures comme clé étrangère de la table des enregistrements des femmes.

id_femme	femme	id_chaussures
1	Carrie	3
2	Samantha	1
3	Charlotte	1
4	Miranda	1
5	Carrie	4
6	Charlotte	2
7	Charlotte	3
8	Charlotte	4
9	Miranda	3
10	Miranda	4

id_chaussures	nom_chaussures
1	sandales Manolo
2	mules
3	tongs Old Navy
4	bottes Prada

Les deux tables sont maintenant reliées par la colonne id_chaussures.

Remarquez les duplicats dans les colonnes femme et nom_chaussures.

Dessinez vous-mêmes les tables, mais cette fois mettez l'id_femme comme clé étrangère de la table des chaussures.

Une fois terminé, dessinez les liens.

id_chaussures	nom_chaussures	id_femme
1	sandales Manolo	3
2	sandales Manolo	2
3	tongs Old Navy	1
4	bottes Prada	1
5	mules	3
6	tongs Old Navy	3
7	bottes Prada	3
8	sandales Manolo	4
9	tongs Old Navy	4
10	bottes Prada	4

id_femme	femme
1	Carrie
2	Samantha
3	Charlotte
4	Miranda

Patterns de données : nous avons besoin d'une table de jonction

Comme vous venez de vous en apercevoir, ajouter l'une ou l'autre des clés primaires à l'autre table en tant que clé étrangère nous donne des données dupliquées dans notre table. Remarquez le nombre de fois où les noms des femmes apparaissent. Nous ne devrions les voir qu'une fois.

Il nous faut une table qui s'intercale entre ces deux tables plusieurs-à-plusieurs et simplifie les relations en un-à-plusieurs. Cette table contiendra toutes les valeurs `id_femme` ainsi que les valeurs `id_chaussures`. Nous avons besoin de ce qu'on appelle une **table de jonction**, qui va contenir les colonnes des clés primaires des deux tables que nous voulons relier.

Relier ces deux tables directement ne peut pas faire l'affaire parce que nous nous retrouvons avec des données dupliquées à cause de leurs relations plusieurs-à-plusieurs.

id_femme	femme
1	Carrie
2	Samantha
3	Charlotte
4	Miranda

plusieurs-à-plusieurs

id_chaussures	nom_chaussures
1	sandales Manolo
2	mules
3	tongs Old Navy
4	bottes Prada

Prenez la clé primaire d'ici...

... et la clé primaire d'ici...

... et placez les toutes les deux dans une table de jonction.

un-à-plusieurs

un-à-plusieurs

id_femme	id_chaussures
1	3
1	4
2	1
3	1
3	2
3	3
3	4
4	1
4	3
4	4

La table de jonction contient les clés primaires des deux tables que vous voulez relier.

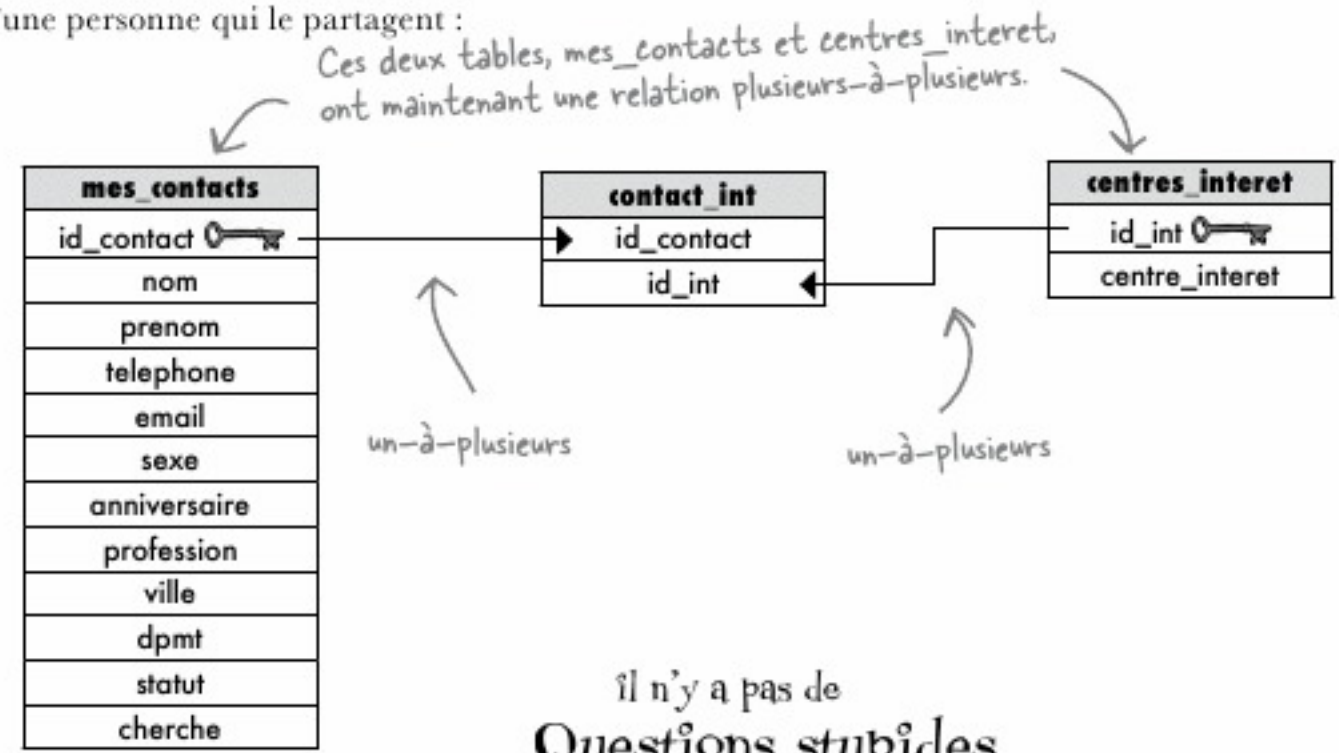
Ensuite il faut relier les colonnes des clés primaires de chacune des deux tables d'origine avec les colonnes correspondantes dans la table de jonction.

Plusieurs-à-plusieurs : une table de jonction contient une clé de chaque table.

Patterns de données : plusieurs-à-plusieurs

Vous connaissez maintenant le secret de la relation **plusieurs-à-plusieurs**. Elle est généralement créée par ***deux relations un-à-plusieurs***, avec une ***table de jonction entre elles***. Nous devons associer **UNE** personne de la table `mes_contacts` avec **PLUSIEURS** centres d'intérêt de notre nouvelle table `centres_interet`. Mais chacune des valeurs de centres_interet peut aussi concerner plus d'une personne, cette relation entre donc dans le pattern **plusieurs-à-plusieurs**.

La colonne `centres_interet` peut être convertie en une relation plusieurs-à-plusieurs en utilisant ce schéma. Chaque personne peut avoir plus d'un centre d'intérêt et pour chaque centre d'intérêt, il peut y avoir plus d'une personne qui le partagent :



il n'y a pas de Questions stupides

Q: Dois-je créer la table du milieu quand j'ai des relations plusieurs-à-plusieurs ?

R: Oui. Si vous avez une relation plusieurs-à-plusieurs entre deux tables, vous allez obtenir des groupes qui se répètent, enfreignant ainsi la première forme normale. (Un rappel sur la normalisation va arriver dans quelques pages.)

Il n'existe pas de bonne raison d'enfreindre la première forme normale et beaucoup de bonnes raisons de ne pas le faire. La plus importante est que vous aurez beaucoup de difficultés à interroger vos tables avec toutes les données répétitives.

Q: Quel est l'avantage de modifier ma table de cette façon ? Je peux tout simplement placer tous les centres d'intérêt dans une table avec id_contact et nom_centre_int. J'aurais des répétitions, mais à part cela, pourquoi pas ?

R: Vous y verrez sans le moindre doute un avantage quand vous commencerez à interroger des tables multiples avec des jointures au prochain chapitre. Cela peut aussi vous aider, selon la façon dont vous utilisez vos données. Vous pouvez avoir une table qui vous intéresse plus pour cette liaison plusieurs-à-plusieurs que pour les données des deux autres tables.

Q: Mais si les répétitions ne me dérangent *toujours* pas ?

R: Joindre des tables aide à protéger l'intégrité de vos données. Si vous devez supprimer quelqu'un de `mes_contacts`, vous ne touchez jamais à la table centres_interet, seulement à la table `contact_int`. Sans la table séparée, vous pourriez accidentellement enlever les mauvais enregistrements. C'est plus sûr de cette façon.

Et quand on en vient à la mise à jour des infos, c'est aussi sympa. Supposez que vous orthographiez mal un nom de passe-temps étrange, comme « clavissophilie ». Quand vous le corrigez, il suffit de modifier une ligne de la table centres_interet, vous n'avez pas à toucher aux tables `contact_int` ou `mes_contacts`.

NOMMEZ CETTE RELATION

Dans chacune des tables partielles ci-dessous, décidez si les colonnes entourées sont mieux représentées par une relation un-à-plusieurs ou plusieurs-à-plusieurs.

(Rappelez-vous que si c'est un-à-plusieurs ou plusieurs-à-plusieurs, la colonne doit être enlevée de la table et reliée avec un champ ID.)

COLONNE	RELATION
evaluation_crepe type_crepe note	. .....
info_clowns id_clown activites date	. .....
mes_contacts id_contact dpmt	. .....
centres_interet	. .....
livres id_livre auteurs	. .....
editeur	. .....
info_poissons id_prise espece	. .....
dpmt	. .....

NOMMEZ CETTE RELATION - SOLUTION

Dans chacune des tables partielles ci-dessous, décidez si les colonnes entourées sont mieux représentées par une relation un-à-plusieurs ou plusieurs-à-plusieurs.

(Rappelez-vous que si c'est un-à-plusieurs ou plusieurs-à-plusieurs, la colonne doit être enlevée de la table et reliée avec un champ ID.)

COLONNE	RELATION
evaluation_crepe type_crepe note	un-à-plusieurs
info_clowns id_clown activites date	plusieurs-à-plusieurs
mes_contacts id_contact dpmt centres_interet	un-à-plusieurs plusieurs-à-plusieurs
livres id_livre auteurs éditeur	Celle-ci est difficile, mais puisqu'un livre peut avoir plus d'un auteur, c'est plusieurs-à-plusieurs. plusieurs-à-plusieurs un-à-plusieurs
info_poissons id_prise espece dpmt	un-à-plusieurs un-à-plusieurs

Patterns de données : arranger liste_greg

Presque. Maintenant que vous connaissez les patterns de données, nous sommes presque prêt à repenser liste_greg.

Nous savons que la colonne centres_interet peut être modifiée en une relation un-à-plusieurs avec une autre table. Nous devons également arranger la colonne cherche de la même façon. Ces modifications vont nous faire passer à la ***première forme normale****.

Mais nous ne pouvons pas simplement nous arrêter à la première forme normale. Nous devons normaliser davantage. Plus nous normalisons maintenant, plus il vous sera facile de récupérer vos données avec des requêtes et, dans le prochain chapitre, des jointures. Avant de créer un nouveau schéma pour `liste_greg`, faisons un détour pour en apprendre plus sur les niveaux de normalisation.

mes_contacts
id_contact
nom
prenom
telephone
email
sexe
anniversaire
profession
ville
dpmt
statut
centres_interet
cherche

* Vous vous sentez peut-être obligé de retourner quelques chapitres en arrière pour vous rafraîchir la mémoire sur la première forme normale. Ce n'est pas la peine, nous en parlons page suivante.

Pas à la première forme normale

Nous avons parlé de la première forme normale. Revenons-y avant de pousser notre normalisation encore plus loin, vers la deuxième et même la troisième forme normale.

Mais avant de pouvoir en arriver là, récapitulons ce qu'il faut pour passer une table en 1FN.

Première forme normale ou 1FN :
Règle 1 : Les colonnes ne contiennent que des valeurs atomiques
Règle 2 : Pas de répétition de groupes de données

Les tables ci-dessous ne sont pas à la première forme normale. Remarquez que la seonde table a des colonnes couleur supplémentaires, mais les couleurs elles-mêmes continuent à se répéter à chaque ligne de la nouvelle table :

Pas en 1FN

id_jouet	jouet	couleurs
5	raquette	blanc, jaune, bleu
6	frisbee	vert, jaune
9	cerf-volant	rouge, bleu, vert
12	yoyo	blanc, jaune

Pour être atomique, la colonne couleurs ne doit contenir qu'une de ces couleurs, au lieu de 2 ou 3 par enregistrement.

Toujours pas en 1FN

id_jouet	jouet	couleur1	couleur2	couleur3
5	raquette	blanc	jaune	bleu
6	frisbee	vert	jaune	
9	cerf-volant	rouge	bleu	vert
12	yoyo	blanc	jaune	

Cette table n'est toujours pas en 1FN parce que les colonnes elles-mêmes contiennent la même catégorie de données, toutes des VARCHAR avec la couleur du jouet.

Enfin en 1FN

Voyons ce que nous avons fait ici.

En 1FN

Clé primaire.

id_jouet	jouet
5	raquette
6	frisbee
9	cerf-volant
12	yoyo

Clé étrangère.

id_jouet	couleur
5	blanc
5	jaune
5	bleu
6	vert
6	jaune
9	rouge
9	bleu
9	vert
12	blanc
12	jaune

Rien n'est répété dans une colonne. Nous avons une couleur par ligne et toutes nos lignes sont uniques.

Nous savons maintenant que ces tables sont jointes par la clé étrangère, nous n'avons plus besoin de dessiner des lignes partout.

Nous pouvons utiliser les valeurs id_jouet et couleur ensemble pour former une clé primaire unique.

Si nous ajoutons `id_jouet` dans une table séparée comme clé étrangère, cela fonctionne parce que les valeurs qu'elle contient ne doivent pas être uniques. Si nous ajoutons également les valeurs de couleur à cette table, **toutes les lignes sont uniques** parce que chaque couleur PLUS chaque `id_jouet` ensemble forment une ***combinaison unique***.

Non. Une clé composée de deux colonnes ou plus s'appelle une clé composite.

Voyons comment ces clés fonctionnent dans d'autres tables.

Les clés composites utilisent plusieurs colonnes

Jusqu'ici nous avons parlé de la façon dont les données d'une table *sont liées à d'autres tables* (un-à-un, un-à-plusieurs). Nous n'avons pas pris en compte **la façon dont les colonnes d'une table *sont liées les unes aux autres***. Comprendre cela est la clé pour comprendre la deuxième et la troisième forme normale.

Et une fois que nous comprenons ces formes, nous pouvons créer des schémas de base de données qui vont grandement faciliter l'interrogation de plusieurs tables.

Vous aurez besoin de tables bien conçues quand nous arriverons aux jointures dans le prochain chapitre !

Qu'*est* donc exactement une clé composite ?

UNE CLÉ COMPOSITE est une CLÉ PRIMAIRE composée de plusieurs colonnes, créant une clé unique.

Prenez cette table de super-héros. Elle n'a pas de clé unique, mais nous pouvons créer une clé primaire composite à partir des colonnes `nom` et `pouvoir`. Même s'il y a des noms et des pouvoirs dupliqués, une fois placés ensemble, la paire qu'ils forment crée une valeur unique.

Nous pouvons créer cette table et désigner ces deux champs comme clé primaire composite. Nous supposons que nous n'aurons jamais les mêmes noms et pouvoirs et que cela sera donc unique.

super_heros

nom	pouvoir	faiblesse
Super Homme Poubelle	Nettoie rapidement	eau de javel
Le Courtier	Fabrique de l'argent à partir de rien	NULL
Super Mec	Vole	oiseaux
Super Serveur	N'oublie jamais une commande	insectes
L'homme sale	Crée des tempêtes de poussière	eau de javel
Super Mec	Super force	l'autre Super Mec
Femme Furieuse	Se met vraiment très en colère	NULL
Le Crapaud	Langue de la justice	insectes
Bibliothécaire	Peut trouver n'importe quoi	NULL
Jeune Dinde	Vole	NULL
Bonhomme Bâton	Remplace les humains	le pendu

Petit Bonhomme Bâton,
Fait ce à quoi l'homme n'est pas bon
Il ne manque qu'un numéro 2
Pour dire quoi faire au petit bâton
Libérez votre imagination
Pour dessiner en moins de deux
Votre propre Bonhomme Bâton !

Même les super-héros peuvent être dépendants

Nos super-héros n'ont pas chômé ! Voici la table `super_heros` mise à jour. Nous sommes en 1FN, mais il y a un autre problème.

Voyez-vous que la colonne `initiales` contient les premières lettres de la valeur nom de la colonne `nom` ? Qu'arriverait-il si un super-héros changeait de nom ?

Exactement. La colonne initiales changerait aussi. La colonne `initiales` est dite **fonctionnellement dépendante** de la colonne `nom`.

Si les données d'une colonne doivent changer quand celles d'une autre colonne sont modifiées, la première colonne est fonctionnellement dépendante de la seconde.

Voici nos deux noms identiques, avec la colonne pouvoir ajoutée pour créer une clé primaire véritablement unique.

super_heros

nom	pouvoir	faiblesse	ville	pays	pire_ennemi	initiales
Super Homme Poubelle	Nettoie rapidement	eau de javel	Gotham	Etats-Unis	Verminator	SHP
Le Courtier	Fabrique de l'argent à partir de rien	NULL	New York	Etats-Unis	Monsieur Fisc	LC
Super Mec	Vole	oiseaux	Metropolis	Etats-Unis	Super Type	SM
Super Serveur	N'oublie jamais une commande	insectes	Paris	France	La Fille Estomac	SS
L'homme sale	Crée des tempêtes de poussière	eau de javel	Tulsa	Etats-Unis	Aspirateur	HS
Super Mec	Super force	aluminium	Metropolis	Etats-Unis	Le Méchant	SM
Femme Furieuse	Se met vraiment très en colère	NULL	Rome	Italie	Le Thérapeute	FF
Le Crapaud	Langue de la justice	insectes	Londres		Le Héron	C
Bibliothécaire	Peut trouver n'importe quoi	enfants	Springfield	Etats-Unis	Monsieur Chaos	B
Jeune Dinde	Vole	NULL		Etats-Unis	Machine à Coudre	JD
Bonhomme Bâton	Remplace les humains	le pendu	Londres		Monsieur Gomme	BB

À vos crayons !

Vous savez maintenant que la colonne initiales est dépendante de la colonne nom de la table super_heros. Voyez-vous d'autres dépendances similaires ? Notez-les ici.

. .....

. .....

. .....

. .....

À vos crayons !
Solution

Vous savez maintenant que la colonne initiales est dépendante de la colonne nom de la table super_heros. Voyez-vous d'autres dépendances similaires ? Notez-les ici.

initiales dépend du nom

faiblesse dépend du nom

pire_ennemi dépend du nom

ville dépend du pays

Cela ne mentionne pas de quelle table viennent les colonnes, ce qui aura une importance quand vous ajouterez d'autres tables. Il existe une façon abrégée d'indiquer ces dépendances et les tables d'où elles viennent.

Notations abrégées

On peut décrire une dépendance fonctionnelle rapidement en écrivant ceci :

T.x —>; T.y ← En termes techniques, on appelle ceci une notation abrégée.

Ce qui peut se lire « dans la table relationnelle appelée T, la colonne y est fonctionnellement dépendante de la colonne x ». Vous les lisez en fait de droite à gauche pour voir ce qui est fonctionnellement dépendant de quoi.

Voyons cela appliqué à nos super-héros :

super_heros.nom —>; super_heros.initiales

« Dans la table relationnelle `super_heros`, la colonne `initiales` est fonctionnellement dépendante de la colonne `nom`. »

super_heros.nom —>; super_heros.faiblesse

« Dans la table relationnelle `super_heros`, la colonne `faiblesse` est fonctionnellement dépendante de la colonne `nom`. »

super_heros.nom —>; super_heros.pire_ennemi

« Dans la table relationnelle `super_heros`, la colonne `pire_ennemi` est fonctionnellement dépendante de la colonne `nom`. »

super_heros.pays —>; super_heros.ville

« Dans la table relationnelle `super_heros`, la colonne `ville` est fonctionnellement dépendante de la colonne `pays`. »

Dépendances de Super-héros

Donc, si notre super-héros venait à changer son nom, la colonne initiales devrait changer aussi, ce qui la rend **dépendante** de la colonne nom.

Si notre pire_ennemi décide de déménager son repaire dans une nouvelle ville, son adresse change, mais rien d'autre. Cela rend la colonne `ville_pire_ennemi` de la table ci-dessous complètement **indépendante**.

Une **colonne dépendante** est une colonne contenant des données qui ***peuvent changer si une autre colonne change***. Les colonnes **non-dépendantes** ***fonctionnent seules***.

Dépendance fonctionnelle partielle

Une **dépendance fonctionnelle partielle** signifie qu'une colonne non-clé est dépendante de certaines, mais pas de toutes les colonnes d'une clé primaire composite.

Dans notre table de super-héros, la colonne `initiales` est ***partiellement dépendante*** de `nom`, parce que si le nom du super-héros change, la valeur initiales aussi, mais si le pouvoir change et pas le nom, les initiales de notre super-héros restent les mêmes.

Ensemble, nom et pouvoir forment la clé primaire composite.

Intiales dépend de nom, mais pas de pouvoir, cette table contient donc une dépendance fonctionnelle partielle.

super_heros

nom	pouvoir	faiblesse	ville	initiales	id_pire_ennemi	ville_pire_ennemi
Super Homme Poubelle	Nettoie rapidement	eau de javel	Gotham	SHP	4	Gotham
Le Courtier	Fabrique de l'argent à partir de rien	NULL	New York	LC	8	Newark
Super Mec	Vole	oiseaux	Metropolis	SM	5	Metropolis
Super Serveur	N'oublie jamais une commande	insectes	Paris	SS	1	Paris
L'homme sale	Crée des tempêtes de poussière	eau de javel	Tulsa	HS	2	Kansas City
Super Mec	Super force	aluminium	Metropolis	SM	7	Gotham
Femme Furieuse	Se met vraiment très en colère	NULL	Rome	FF	10	Rome
Le Crapaud	Langue de la justice	insectes	Londres	C	16	Bath
Bibliothécaire	Peut trouver n'importe quoi	enfants	Springfield	B	3	Louisville
Jeune Dinde	Vole	NULL	Minneapolis	JD	9	Minneapolis
Super Bâton	Remplace les humains	le pendu	Londres	SB	33	Borrowdale

Dépendance fonctionnelle transitive

Il vous faut également prendre en considération la façon dont chaque colonne non-clé est reliée aux autres. Si un pire-ennemi déménage dans une autre ville, cela ne change pas son `id_pire_ennemi`.

L'id_pire_ennemi de Super Homme Poubelle n'a pas changé, même s'il a déménagé à Kansas City.

nom	id_pire_ennemi	ville_pire_ennemi
Super Homme Poubelle	4	Kansas City
Le Courtier	8	Newark
Super Mec	5	Metropolis
Super Serveur	1	Paris
L'homme sale	2	Kansas City

Si modifier une colonne non-clé peut entraîner des changements dans l'une des autres colonnes, vous avez une dépendance transitive.

Supposez qu'un super-héros change de `pire_ennemi`. L'`id_pire_ennemi` change et cela ***peut*** changer la `ville_pire_ennemi`.

Si modifier une des colonnes non-clés peut provoquer un changement dans une des autres colonnes, vous avez une ***dépendance transitive***.

Si nous mettons à jour l'id_pire_ennemi cela modifie la valeur de la colonne ville_pire_ennemi.

nom	id_pire_ennemi	ville_pire_ennemi
Super Homme Poubelle	2	Kansas City
Le Courtier	8	Newark
Super Mec	5	Metropolis
Super Serveur	1	Paris
L'homme sale	2	Kansas City

Cela s'appelle une dépendance fonctionnelle transitive parce que la colonne non-clé ville_pire_ennemi est liée à id_pire_ennemi, qui est une autre colonne non-clé.

Dépendance fonctionnelle transitive : quand une colonne non-clé est liée à une autre colonne non-clé.

Regardez cette table répertoriant des titres de livres. id_ed identifie l'éditeur. ville_pub est la ville où le livre a été publié.

auteur	titre	copyright	id_ed	ville_pub
José Bové	Facile d'être vert	1930	2	New York
Dorothée	Je hais les dessins animés	1968	5	Paris
Lassie	À l'aide Timmy!	1950	3	Wadi Rum
Timmy	Lassie, calme-toi	1951	1	New York

Notez ce que va arriver à la valeur de la colonne copyright si le titre du livre de la troisième ligne est modifié en : 'À l'aide Timmy ! Je suis coincé dans un puits'

Si le titre change, la valeur de copyright change aussi.

Le copyright dépend du titre, sa valeur va donc changer.

Qu'arrivera-t-il à la valeur de la colonne copyright si l'auteur du livre de la troisième ligne est modifié en 'Rin Tin Tin', mais que le titre reste le même ?

Qu'arriverait-il à 'Facile d'être vert' si nous changions sa valeur id_ed en 1 ?

Qu'arriverait-il à la valeur id_ed de 'Je hais les dessins animés' si son éditeur déménageait à Sébastopol ?

Qu'arriverait-il à la valeur ville_pub de 'Je hais les dessins animés' si nous changions sa valeur id_ed en 1 ?

Solutions des exercices

Regardez cette table répertoriant des titres de livres. id_ed identifie l'éditeur. ville_pub est la ville où le livre a été publié.

Notez ce que va arriver à la valeur de la colonne copyright si le titre du livre de la troisième ligne est modifié en : 'À l'aide Timmy ! Je suis coincé dans un puits'

Si le titre change, la valeur de copyright change aussi.

Le copyright dépend du titre, sa valeur va donc changer.

Qu'arrivera-t-il à la valeur de la colonne copyright si l'auteur du livre de la troisième ligne est modifié en 'Rin Tin Tin', mais que le titre reste le même ?

Ensemble, auteur et titre forme la clé primaire composite.

Si l'auteur change mais pas le titre, le copyright change.

Le copyright dépend du titre. Il dépend aussi de l'auteur.

auteur	titre	copyright	id_ed	ville_pub
José Bové	Facile d'être vert	1930	2	New York
Dorothée	Je hais les dessins animés	1968	5	Paris
Lassie	À l'aide Timmy!	1950	3	Wadi Rum
Timmy	Lassie, calme-toi	1951	1	New York

Qu'arriverait-il à 'Facile d'être vert' si nous changions sa valeur id_ed en 1 ?

La ville_pub ne changerait pas.

La ville_pub pour id_ed 1 et id_ed 2 est New York, la ville ne changerait donc pas (même si ville_pub est transitivement dépendante d'id_ed).

id_ed est indépendant de la colonne ville_pub, l'id_ed reste donc le même.

Qu'arriverait-il à la valeur id_ed de 'Je hais les dessins animés' si son éditeur déménageait à Sébastopol ?

L'id_ed resterait le même.

Qu'arriverait-il à la valeur ville_pub de 'Je hais les dessins animés' si nous changions sa valeur id_ed en 1 ?

ville_pub est transitivement dépendante d'id_ed donc la valeur ville change.

La ville_pub deviendrait New York.

ville_pub est dépendante de la valeur de la colonne id_ed. Aucune des colonnes n'est une colonne clé, c'est donc une dépendance fonctionnelle transitive.

auteur	titre	copyright	id_ed	ville_pub
José Bové	Facile d'être vert	1930	2	New York
Dorothée	Je hais les dessins animés	1968	5	Paris
Lassie	À l'aide Timmy!	1950	3	Wadi Rum
Timmy	Lassie, calme-toi	1951	1	New York

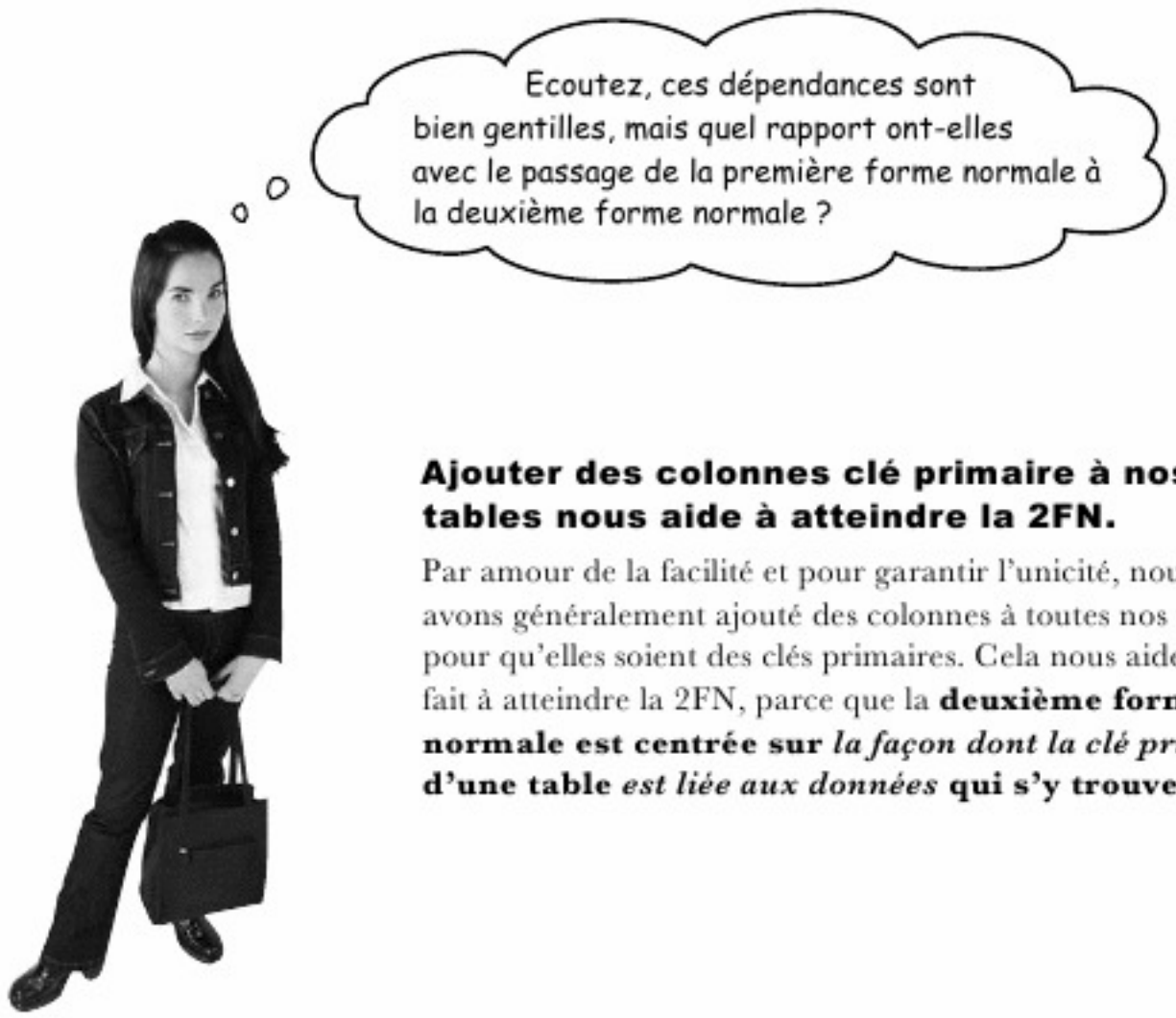
il n'y a pas de
Questions stupides

Q: Existe-t-il un moyen simple pour éviter d'avoir une dépendance fonctionnelle partielle ?

R: Utiliser un champ id comme nous le faisons dans `mes_contacts` vous permet d'éviter complètement le problème. Comme c'est une nouvelle clé qui n'existe que pour indexer cette table, rien n'en dépend.

Q: Donc, à part si je crée des tables de jonction, pourquoi voudrais-je créer une clé composite à partir de colonnes de ma table ? Pourquoi ne pas toujours créer un champ id ?

R: C'est certainement une bonne solution. Mais vous trouverez des arguments convaincants des deux côtés sur internet (en anglais : *synthetic key* vs *natural key*). Vous trouverez aussi des débats enflammés. Nous vous laissons vous faire votre propre opinion sur le sujet. Dans ce livre, nous en resterons principalement aux clés primaires synthétiques uniques pour simplifier notre syntaxe, que vous puissiez apprendre les concepts sans vous enliser dans leur mise en œuvre.



Ajouter des colonnes clé primaire à nos tables nous aide à atteindre la 2FN.

Par amour de la facilité et pour garantir l'unicité, nous avons généralement ajouté des colonnes à toutes nos tables pour qu'elles soient des clés primaires. Cela nous aide en fait à atteindre la 2FN, parce que la **deuxième forme normale est centrée sur *la façon dont la clé primaire* d'une table *est liée aux données* qui s'y trouvent**.

Deuxième forme normale

Prenons deux tables existantes pour créer un inventaire de jouets et nous aider à mieux comprendre comment la deuxième forme normale s'intéresse à la relation entre la clé primaire de la table et les données de la table.

id_jouet	jouet
5	raquette
6	frisbee
9	cerf-volant
12	yoyo

Clé composite.

id_jouet O+	id_magasin O+	couleur	inventaire	adresse_magasin
5	1	blanc	34	23 rue des érables
5	3	jaune	12	100 boulevard du nord
6	1	bleu	5	23 rue des érables
6	2	vert	10	1902 allée de l'ambre
6	4	jaune	24	17 route moulin de pont
9	1	rouge	50	23 rue des érables
9	2	bleu	2	1902 allée de l'ambre
12	2	vert	18	1902 allée de l'ambre
12	4	blanc	28	17 route moulin de pont
9	4	jaune	11	17 route moulin de pont

Il y a beaucoup de répétitions dans cette colonne. Et cela n'a vraiment rien à voir avec l'inventaire ; c'est lié au magasin.

Nous devrions repenser cette colonne également. Elle appartient plus à une table jouet qu'à une table inventaire. Notre id_jouet devrait identifier à la fois le type de jouet ET sa couleur.

inventaire est dépendant des deux colonnes qui forment la clé primaire composite, il n'a donc pas de dépendance fonctionnelle partielle.

Remarquez que `adresse_magasin` se répète quand un jouet est associé avec cet `id_magasin`. Si nous devons modifier l'`adresse_magasin`, nous devons modifier chaque ligne où elle est référencée dans cette table. Avec le temps, plus des lignes seront mises à jour, plus il y aura de possibilités que des erreurs se glissent dans nos données.

Si nous déplaçons l'`adresse_magasin` dans une autre table, nous n'aurons plus qu'un changement à faire.

Nous pourrions déjà être en 2FN...

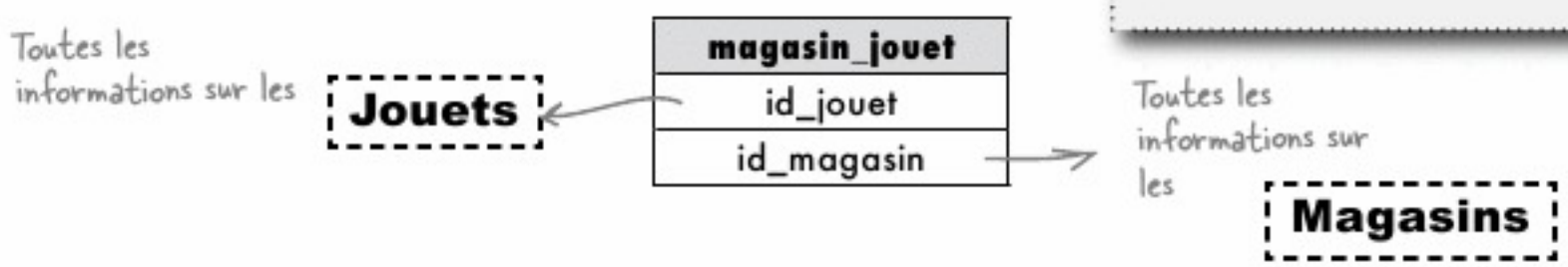
Une table en 1FN est aussi en 2FN si toutes les colonnes de la table font partie de la clé primaire.

Nous pourrions créer une nouvelle table avec une clé primaire composite formée par les colonnes `id_jouet` et `id_magasin`. Alors nous aurions une table avec toutes les informations sur les jouets, une table avec toutes les informations sur les magasins et notre nouvelle table pour les relier.

Votre table 1FN est aussi 2FN si toutes les colonnes de la table font partie de la clé primaire

OU

si elle a une seule colonne clé primaire



Une table en 1FN est aussi en 2FN si elle a une seule colonne clé primaire.

C'est une excellente raison de désigner une colonne id `AUTO_INCREMENT`.

Deuxième forme normale ou 2FN :
Règle 1 : Être en 1FN
Règle 2 : Ne pas avoir de dépendances fonctionnelles partielles.

C'est pourquoi il est l'heure de jouer...

Vous ÊTES la table 2FN sans dépendances fonctionnelles partielles

Votre mission est de devenir une table et de vous purifier de toutes vos dépendances fonctionnelles partielles. Regardez chacune des tables schématisées ci-dessous et barrez les colonnes qui seraient mieux à leur place dans une autre table.

Ces deux-là forment une clé primaire composite unique.

jouet_inventaire
id_jouet
id_magasin

ventes_biscuits
quantite
id_fille
date
nom_fille
cheftaine
total_ventes

chanteurs
id_chanteur
nom
prenom
agence
dpmt_agence

films
id_film
titre
genre
loue_par
a_rendre_le
evaluation

salaire
id_employe
nom
prenom
salaire
directeur
email_employe
date_embauche

races_chiens
race
description
poids_moyen
taille_moyenne
id_club
dpmt_club

À vos crayons !

Réorganisez ces tables en trois tables qui soient toutes en 2FN.

L'une contiendra les infos sur les jouets, une autre les infos sur les magasins et la troisième contiendra l'inventaire et reliera les deux autres. Donnez-leur toutes des noms significatifs.

Enfin, ajoutez ces colonnes supplémentaires aux tables appropriées : ***prix du jouet, numéro de téléphone du magasin, nom du directeur du magasin et poids du jouet***.

id_jouet	jouet
5	raquette
6	frisbee
9	cerf-volant
12	yoyo

id_jouet	id_magasin	couleur	inventaire	adresse_magasin
5	1	blanc	34	23 rue des érables
5	3	jaune	12	100 boulevard du nord
6	1	bleu	5	23 rue des érables
6	2	vert	10	1902 allée de l'ambre
6	4	jaune	24	17 route moulin de pont
9	1	rouge	50	23 rue des érables
9	2	bleu	2	1902 allée de l'ambre
12	2	vert	18	1902 allée de l'ambre
12	4	blanc	28	17 route moulin de pont
9	4	jaune	11	17 route moulin de pont

Vous ÊTES la table 2FN sans dépendances fonctionnelles partielles - solution

Votre mission est de devenir une table et de vous purifier de toutes vos dépendances fonctionnelles partielles. Regardez chacune des tables schématisées ci-dessous et barrez les colonnes qui seraient mieux à leur place dans une autre table.

Ces deux-là forment une clé primaire composite unique.

jouet_inventaire
id_jouet
id_magasin

Clé primaire.

chanteurs
id_chanteur
nom
prenom
agence
dpmt_agence

Même si ces colonnes devraient être un ID tiré d'une table agences (parce que deux agences peuvent avoir le même nom), ce n'est pas une dépendance fonctionnelle partielle.

ventes_biscuits
quantite
id_fille
date
~~nom_fille~~
~~cheftaine~~
~~total_ventes~~

Une fois ces colonnes déplacées, les colonnes restantes peuvent former une clé primaire composite.

Clé primaire.

Ces colonnes n'ont qu'une dépendance fonctionnelle transitive.

films
id_film
titre
genre
loue_par
a_rendre_le
evaluation

Clé primaire.

Même si ces colonnes devraient être enlevées de cette table, elles ne sont pas partiellement dépendantes.

salaire
id_employe
nom
prenom
salaire
directeur
email_employe
date_embauche

races_chiens
race
description
poids_moyen
taille_moyenne
id_club
~~dpmt_club~~

Clé primaire composite.

id_club peut appartenir à cette table (si c'est une relation un-à-un), mais dpmt_club n'a pas sa place ici. Malgré cela, aucune des colonnes n'est partiellement dépendante.

À vos crayons ! Solution

Réorganisez ces tables en trois tables qui soient toutes en 2FN.

L'une contiendra les infos sur les jouets, une autre les infos sur les magasins et la troisième contiendra l'inventaire et reliera les deux autres. Donnez-leur toutes des noms significatifs.

Enfin, ajoutez ces colonnes supplémentaires aux tables appropriées : ***prix du jouet, numéro de téléphone du magasin, nom du directeur du magasin et poids du jouet***.

id_jouet	jouet
5	raquette
6	frisbee
9	cerf-volant
12	yoyo

id_jouet	id_magasin	couleur	inventaire	adresse_magasin
5	1	blanc	34	23 rue des érables
5	3	jaune	12	100 boulevard du nord
6	1	bleu	5	23 rue des érables
6	2	vert	10	1902 allée de l'ambre
6	4	jaune	24	17 route moulin de pont
9	1	rouge	50	23 rue des érables
9	2	bleu	2	1902 allée de l'ambre
12	2	vert	18	1902 allée de l'ambre
12	4	blanc	28	17 route moulin de pont
9	4	jaune	11	17 route moulin de pont

La clé primaire composite est id_jouet et id_magasin.

inventaire_magasins

id_jouet	id_magasin	inventaire
5	1	34
5	3	12
6	1	5
6	2	10
6	4	24
9	1	50
9	2	2
12	2	18
12	4	28
9	4	11

infos_jouets

id_jouet	jouet	couleur	prix	poids
1	raquette	blanc	1.95	0.3
2	raquette	jaune	2.20	0.4
3	frisbee	bleu	1.95	0.3
4	frisbee	vert	3.50	0.5
5	frisbee	jaune	1.50	0.2
6	cerf-volant	rouge	5.75	1.2
7	cerf-volant	bleu	5.75	1.2
8	yoyo	vert	3.15	0.8
9	yoyo	blanc	4.25	0.4
10	cerf-volant	jaune	1.50	0.2

infos_magasins

id_magasin	adresse	telephone	directeur
1	23 rue des érables	0112233445	Jean
2	1902 allée de l'ambre	0198877665	Isabelle
3	100 boulevard du nord	0223345678	Liane
4	17 route moulin de pont	0455443322	Grégoire

Troisième forme normale (enfin)

Puisque nous ajoutons généralement des clés primaires artificielles dans ce livre, passer nos tables à la deuxième forme normale n'est normalement pas un souci pour nous. Toute table avec une **clé primaire artificielle** et sans clé primaire composite est toujours en 2FN.

Mais nous devons nous vérifier que nous somme en 3FN.

Si votre table a une clé primaire artificielle et pas de clé primaire composite, elle est en 2FN

Troisième forme normale ou 3FN :
Règle 1 : Être en 2FN
Règle 2 : Ne pas avoir de dépendances transitives

Vous en souvenez-vous ? Une dépendance fonctionnelle transitive signifie qu'une colonne non-clé est liée à une autre colonne non-clé.

Si modifier l'une des colonnes non-clés peut provoquer un changement dans une autre colonne, vous avez une dépendance transitive.

Considérons ce qui se passe si nous modifions une valeur dans l'une de ces trois colonnes : **`nom_cours`**, **`formateur`** et **`tel_formateur`**.

Nous pouvons ignorer la clé primaire pour 3FN.

cours
id_cours
nom_cours
formateur
tel_formateur

- Si nous changeons le `nom_cours`, ni le `formateur`, ni le `tel_formateur` ne doivent changer.
- Si nous modifions le `tel_formateur`, ni le `formateur`, ni le `nom_cours` ne doivent changer.
- Si nous changeons le `formateur`, le `tel_formateur` va changer. Nous avons trouvé notre dépendance transitive.

Il doit être assez évident maintenant que tel_formateur n'a pas sa place dans cette table si vous voulons qu'elle soit en 3FN.

Alors la table mes_contacts est-elle à la hauteur ?

Elle a bien sûr besoin de quelques modifications. Sur la page ci-dessous, commencez par la table mes_contacts actuelle et dessinez le nouveau schéma de la liste_greg. Montrez les relations entre les clés étrangères avec des lignes et les relations un-à-plusieurs avec des flèches. Indiquez également les clés primaires ou composites.

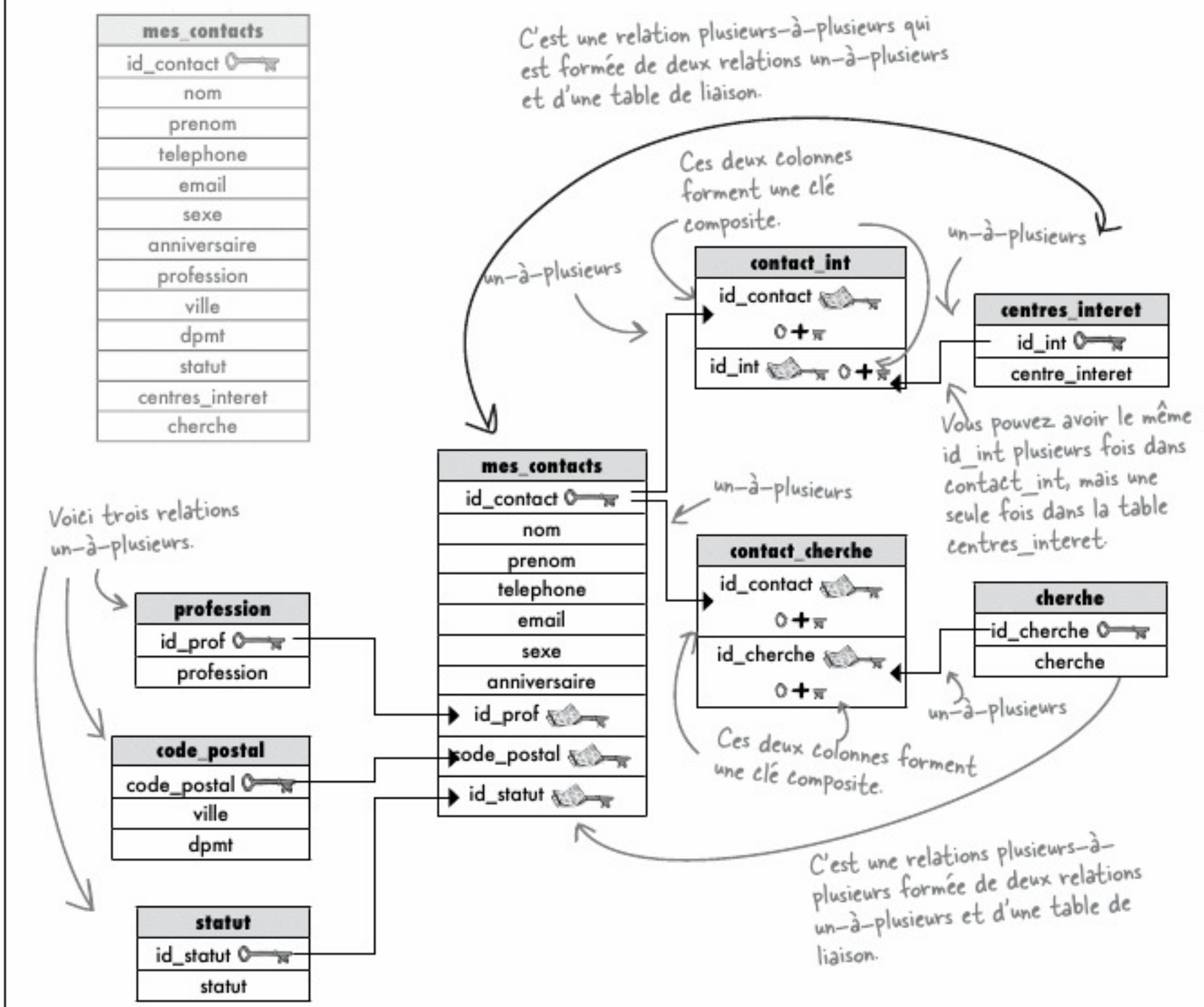
mes_contacts
id_contact
nom
prenom
telephone
email
sexe
anniversaire
profession
ville
dpmt
statut
centres_interet
cherche

Indice : Dans notre version page suivante, nous avons 8 tables. (Nous avons ajouté une colonne pour le code postal. Avant cela, nous en avions 7.)

Solutions des exercices

Alors la table mes_contacts est-elle à la hauteur ?

Elle a bien sûr besoin de quelques modifications. Sur la page ci-dessous, commencez par la table mes_contacts actuelle et dessinez le nouveau schéma de la liste_greg. Montrez les relations entre les clés étrangères avec des lignes et les relations un-à-plusieurs avec des flèches. Indiquez également les clés primaires ou composites.



Et ainsi, Régis (et liste_greg) vécut heureux et eut beaucoup d'enfants

Greg peut trouver la femme idéale pour Régis en utilisant sa base de données fraîchement normalisée. Mieux encore, il peut aussi facilement trouver des correspondances pour d'autres de ses amis, entretenant ainsi le rêve de la Liste de Greg.

Fin

Pas si vite ! Maintenant je dois interroger toutes ces nouvelles tables et les faire correspondre à la main ! Comment puis-je récupérer mes données à présent, avec toutes ces tables, sans écrire des centaines de requêtes ?

C'est là que les jointures entrent en jeu.

Rendez-vous au prochain chapitre...

Boîte à outils SQL

Félicitez-vous, vous en êtes à plus de la moitié du livre. Jetez un œil à tous les termes-clés SQL que vous avez appris dans le chapitre 7. Pour une liste exhaustive des outils du livre, voyez l'annexe C.

Schéma

Une description des données de votre base de données ainsi que de tout autre objet qui s'y rapporte et de la façon dont ils sont tous liés entre eux.

Relation un-à-un

Très exactement une ligne d'une table parente est reliée à une ligne d'une table enfant.

Relation un-à-plusieurs

Une ligne d'une table peut avoir plusieurs lignes correspondantes dans la seconde table, mais la seconde table ne peut avoir qu'une ligne correspondante dans la première.

Relation plusieurs-à-plusieurs

Deux tables sont reliées par une table de jonction, permettant à plusieurs lignes de la première de correspondre à plusieurs lignes de la seconde et inversement.

Première forme normale (1FN)

Les colonnes ne contiennent que des valeurs atomiques et aucun groupe de données se répétant n'est autorisé dans une colonne.

Dépendance fonctionnelle transitive

Cela signifie qu'une colonne non-clé est liée à une autre colonne non-clé.

Deuxième forme normale (2FN)

Votre table doit être en 1FN et ne contenir aucune dépendance fonctionnelle partielle pour être en 2FN.

Troisième forme normale (3FN)

Votre table doit être en 2FN et ne pas avoir de dépendances transitives.

Clé étrangère

C'est une colonne de la table qui référencie la clé primaire d'une autre table.

Clé composite

C'est une clé primaire composée de plusieurs colonnes, ce qui crée une valeur de clé unique.

À vos crayons ! Solution

Utilisez votre ALTER et la fonction SUBSTRING_INDEX pour obtenir ces colonnes. Écrivez autant de requêtes que nécessaire.

id_contact
nom
prenom
telephone
email
sexe
anniversaire
profession
ville
dpmt
statut
centre_interet1
centre_interet2
centre_interet3
centre_interet4
cherche

D'abord vous devez créer les nouvelles colonnes :

```
ALTER TABLE mes_contacts
ADD COLUMN centre_interet1 VARCHAR(50),
ADD COLUMN centre_interet2 VARCHAR(50),
ADD COLUMN centre_interet3 VARCHAR(50),
ADD COLUMN centre_interet4 VARCHAR(50);
```

Puis vous devez déplacer le premier centre d'intérêt dans la nouvelle colonne centre_interet1. Vous pouvez le faire avec :

```
UPDATE mes_contacts
SET centre_interet1 = SUBSTRING_INDEX(centres_interet, ',',
1);
```

Ensuite nous devons enlever le premier centre d'intérêt du champ centres_interet puisqu'il est conservé dans centre_interet1. Nous enlevons tout jusqu'à la première virgule avec une fonction de chaîne :

TRIM enlève l'espace laissé devant la chaîne après que nous ayons enlevé tout ce qu'il y avait devant la virgule.

RIGHT renvoie une partie de la colonne centres_interet en partant de la droite.

```
UPDATE mes_contacts SET centres_interet = TRIM(RIGHT(centres_
interet, (LENGTH(centres_interet)-LENGTH(centre_interet1) -
1)));
```

Cette partie d'apparence effrayante détermine la longueur de la colonne centres_interet dont nous avons besoin. Elle prend la longueur totale de la colonne centres_interet et soustrait la longueur de la partie que nous avons déplacée dans centre_interet1. Puis nous soustrayons un de plus pour commencer après la virgule.

Et nous répétons maintenant ces étapes pour les autres colonnes de centres d'intérêt :

```
UPDATE mes_contacts SET centre_interet2 = SUBSTRING_INDEX(centres_interet, ',', 1);
UPDATE mes_contacts SET centres_interet = TRIM(RIGHT(centres_interet,
(LENGTH(centres_interet)-LENGTH(centre_interet2) - 1)));
UPDATE mes_contacts SET centre_interet3 = SUBSTRING_INDEX(centres_interet, ',', 1);
UPDATE mes_contacts SET centres_interet = TRIM(RIGHT(centres_interet,
(LENGTH(centres_interet)-LENGTH(centre_interet3) - 1)));
```

Pour la dernière colonne, il ne nous reste plus qu'une seule valeur :

```
UPDATE mes_contacts SET centre_interet4 = centres_interet;
```

Maintenant nous pouvons jeter toute la colonne centres_interet avec un drop. Nous aurions aussi pu la renommer centre_interet4 et ne pas ajouter le dernier ADD COLUMN (en supposant que nous n'ayons que quatre centres d'intérêt).

Solutions des exercices

De la page 286.

Écrivez une requête pour Régis sans utiliser la colonne centres_interet.

```
SELECT * FROM mes_contacts
WHERE sexe = 'F'
AND statut = 'célibataire
AND dpmt='37'
AND cherche LIKE '%H célibataire%'
AND anniversaire > '1950-03-20'
AND anniversaire < '1960-03-20';
```

C'est pour l'essentiel la même requête que celle que Greg a utilisé pour Nico, à part qu'il a laissé les centres_interet de côté.

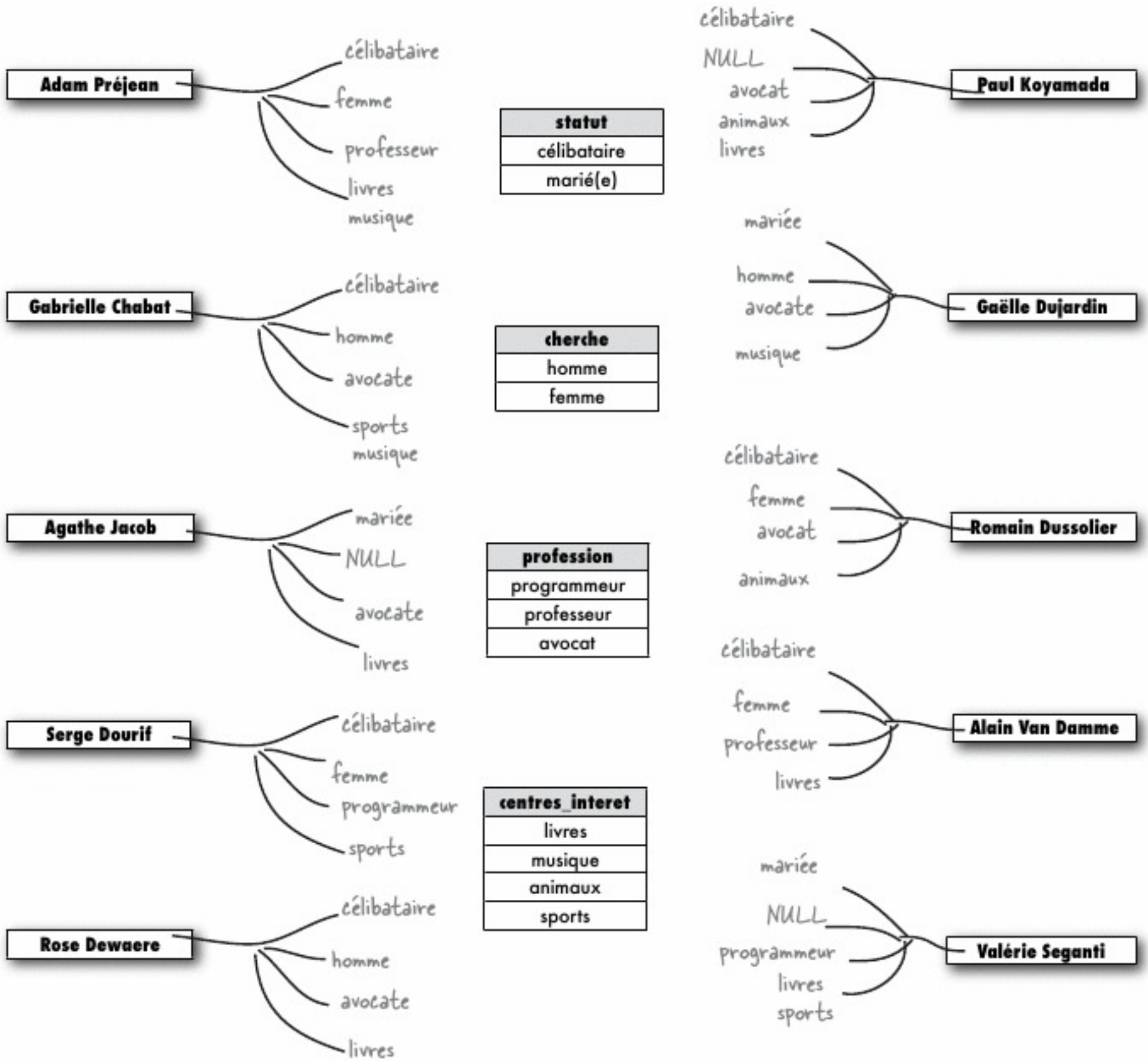
8 jointures et opérations multi-table

Est-ce qu'on ne pourrait pas s'entendre ?

Bienvenue dans un monde multi-table. C'est merveilleux d'avoir **plus d'une table** dans votre base de données, mais vous devez découvrir de ***nouveaux outils et techniques*** pour travailler avec elles. Les tables multiples amènent de la confusion, vous avez donc besoin d'**alias** pour que vos tables restent claires. Et les **jointures** vous aident à relier vos tables, pour que vous puissiez récupérer toutes les données que vous avez éparpillées. Préparez-vous, il est temps de **reprendre le contrôle de votre base de données**.

Nous nous répétons toujours, nous nous répétons...

Greg a remarqué que les mêmes valeurs de **statut**, **profession**, **centres_interet** et **cherche** réapparaissent régulièrement.



Prépeupler vos tables

Ces nombreuses valeurs dupliquées vont faciliter la prépopulation des tables `statut`, `profession`, `centres_interet` et `cherche`. Greg veut charger dans ces quatre tables les valeurs de son ancienne table `mes_contacts`.

Pour commencer, il doit interroger sa table pour savoir ce qui s'y trouve déjà. Mais il ne veut pas obtenir une énorme liste de valeurs dupliquées.

À vos crayons !

Écrivez des requêtes pour récupérer les valeurs statut, profession, centres_interet et cherche de l'ancienne table mes_contacts, sans faire de duplicats. Vous voudrez peut-être retourner voir le problème de vente de biscuits des Jeannettes dans le chapitre 6.

À vos crayons ! Solution

Écrivez des requêtes pour récupérer les valeurs statut, profession, centres_interet et cherche de l'ancienne table mes_contacts, sans faire de duplicats. Vous voudrez peut-être retourner voir le problème de vente de biscuits des Jeannettes dans le chapitre 6.

```
SELECT statut FROM mes_contacts
GROUP BY statut
ORDER BY statut;
```

```
SELECT profession FROM mes_contacts
GROUP BY profession
ORDER BY profession;
```

Utiliser GROUP BY combine les duplicats en une seule valeur pour chaque groupe.

Puis utiliser ORDER BY nous donne la liste dans l'ordre alphabétique.

Si vous ne les mettez pas dans cet ordre, vous obtenez une erreur. ORDER BY doit toujours être en dernier.

```
SELECT cherche FROM mes_contacts
GROUP BY cherche
ORDER BY cherche;
```

```
SELECT centres_interet
FROM mes_contacts
GROUP BY centres_interet
ORDER BY centres_interet;
```

Mais cette requête ne fonctionne pas pour la colonne centres_interet. Nous avons plusieurs valeurs dans celle-là, vous vous souvenez ?

Nous ne pouvons pas faire un simple SELECT pour avoir la colonne centres_interet.

L'utilisation de cette instruction SELECT pour la colonne centres_interet ne va pas fonctionner puisque l'on y trouve des valeurs comme celles-ci :

centres_interet
livres, sport
musique, animaux, livres
animaux, livres
sport, musique

Nous avons le blues de « la table difficile à normaliser »

Oh Marie, si tu savais, notre conception non-normalisée nous fait vraiment souffrir. Il n'existe simplement pas de moyen évident de sortir ces valeurs de la colonne centres_interet pour les voir une par une.

Nous devons passer de ça

centres_interet
premier, deuxième, troisième, quatrième

Notre colonne de la table mes_contacts

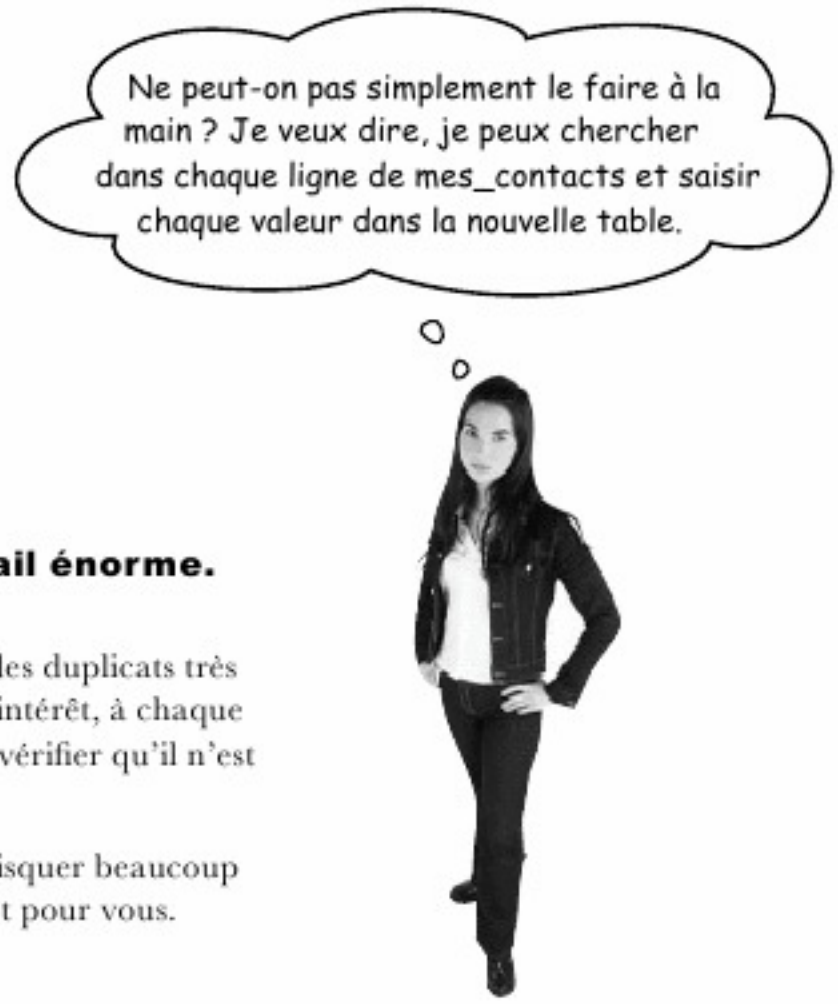
à ça

centres_interet
premier
deuxième
troisième
quatrième

Une colonne de notre nouvelle table centres_interet.

MUSCLEZ VOS NEURONES

Comment pouvons-nous faire passer ces valeurs multiples dans une colonne de la table centres_interet ?



D'abord, c'est une quantité de travail énorme. Imaginez des milliers de lignes.

De plus, le faire à la main rendrait la détection des duplicats très difficile. Si vous avez des centaines de centres d'intérêt, à chaque fois que vous en entrez un nouveau, vous devez vérifier qu'il n'est pas déjà saisi.

Plutôt que de faire tout ce travail pénible et de risquer beaucoup d'erreurs de typo, laissez SQL faire le sale boulot pour vous.

La colonne spéciale centres_interet

Ajouter quatre nouvelles colonnes à `mes_contacts` pour stocker les valeurs temporairement pendant qu'on les sépare est une méthode assez directe. Puis nous pouvons nous débarrasser de ces colonnes quand nous avons fini.

À vos crayons !

Vous savez déjà comment ALTERer des tables. Vous devez ALTERer mes_contacts pour obtenir quatre nouvelles colonnes. Appelez-les centre_interet1, centre_interet2, centre_interet3 et centre_interet4.

→ Réponses page 378.

Voici ce à quoi ressemblent centres_interet et les nouvelles colonnes centre_interet de mes_contacts maintenant que vous avez exécuté `ALTER`.

centres_interet	centre_interet1	centre_interet2	centre_interet3	centre_interet4
premier, deuxième, troisième, quatrième				

Nous pouvons facilement copier le premier centre d'intérêt et le placer dans la nouvelle colonne centre_`interet1` avec notre fonction `SUBSTRING_INDEX` du chapitre 5 :

```
UPDATE mes_contacts
SET centre_interet1 = SUBSTRING_INDEX(centres_interet, ',', 1);
```

Le nom de notre colonne

Le caractère à chercher, une virgule

... on cherche la première virgule.

Exécutez cela et nous obtenons :

centres_interet	centre_interet1	centre_interet2	centre_interet3	centre_interet4
premier, deuxième, troisième, quatrième	premier			

Garder l'intérêt

Maintenant la partie complexe : nous allons utiliser une autre fonction de sous-chaîne pour enlever de la colonne `centres_interet` les données que nous venons de déplacer dans la colonne `centre_interet1`. Puis nous pouvons remplir le reste des colonnes des centres d'intérêt de la même façon.

centres_interet	centre_interet1	centre_interet2	centre_interet3	centre_interet4
premier, deuxième, troisième, quatrième	premier			

Nous allons enlever de la colonne centres_interet le premier centre d'intérêt, la virgule qui le suit et l'espace qui suit la virgule.

Nous utiliserons une fonction `SUBSTR` qui va récupérer la chaîne de la colonne centres_interet et en renvoyer une partie. Nous lui indiquons de renvoyer la chaîne sans la partie que nous venons de mettre dans centre_interet1 et deux caractères de plus (pour la virgule et l'espace).

Traduction : Change la valeur de la colonne centres_interet pour qu'elle devienne ce qu'elle est maintenant sans la partie que nous avons mis dans centre_interet1, la virgule et l'espace.

La longueur du texte dans le champ centre_interet1...

...plus 2 caractères : un pour la virgule, un pour l'espace.

```
UPDATE mes_contacts
SET centres_interet =
SUBSTR(centres_interet, LENGTH(centre_interet1)+2 );
```

SUBSTR renvoie une partie de la chaîne d'origine de cette colonne. Il prend la chaîne, en coupe la première partie telle que définie dans les parenthèses et renvoie la seconde partie.

Length (la longueur) renvoie un nombre correspondant à la longueur de la chaîne se trouvant dans les parenthèses qui le suivent.

Dans notre exemple la chaîne 'premier' fait sept caractères de long.

Vous souvenez-vous que certaines fonctions sont différentes selon la version de SQL que vous utilisez ? Hé bien celle-ci en fait partie. Consultez une référence vraiment utile, comme SQL en concentré

Donc dans notre exemple, le nombre renvoyé par LENGTH est 7+2, soit 9, qui est le nombre de caractères qui vont être enlevés de la première des chaînes dans l'ancienne colonne des centres d'intérêt.

UPDATE met à jour tous vos centres d'intérêt

Après avoir exécuté cette instruction UPDATE, notre table ressemble à ceci.
Mais nous n'avons pas encore fini. Nous devons faire la même chose pour les colonnes `centre_interet2`, `centre_interet3` et `centre_interet4`.

centres_interet	centre_interet1	centre_interet2	centre_interet3	centre_interet4
deuxième, troisième, quatrième	premier			

À vos crayons ! Solution

Complétez les blancs pour terminer l'instruction update de Greg. Nous avons ajouté quelques remarques pour vous aider.

Indice : La colonne centres_interet va changer à chaque fois parce que la valeur de la chaîne dans la colonne centres_interet est raccourcie par la fonction SUBSTR.

```
UPDATE mes_contacts SET
centre_interet1 = SUBSTRING_INDEX(centres_interet, ',', 1),
centres_interet = SUBSTR(centres_interet, LENGTH(centre_interet1)+2),
centre_interet2 = SUBSTRING_INDEX( ..................... ),
centres_interet = SUBSTR( ..................................... ),
centre_interet3 = SUBSTRING_INDEX( ..................... ),
centres_interet = SUBSTR( ..................................... ),
centre_interet4 = ..................................
```

Après avoir enlevé les trois premiers centres d'intérêt de la colonne centres_interet, il ne reste que le quatrième centre d'intérêt. Que faut-il faire ici ?

Complétez avec ce qui se trouve dans chaque colonne après cette longue commande.

centres_interet	centre_interet1	centre_interet2	centre_interet3	centre_interet4
deuxième, troisième, quatrième	premier			

Réponses page 378.

Récupérer tous les centres d'intérêt

Nous avons enfin séparé tous nos centres d'intérêt. Nous pouvons les récupérer avec de simples instructions SELECT, mais pas tous en même temps. Et puisqu'ils sont dans quatre colonnes, nous ne pouvons pas facilement tous les extraire dans un seul ensemble de résultats. Si nous essayons, nous obtenons :

```
Fichier Edition Fenêtre Aide TropDeColonnes

> SELECT centre_interet1, centre_interet2, centre_interet3, centre_
interet4 FROM mes_contacts;

+----------------+----------------+----------------+----------------+
| centre_interet1 | centre_interet2 | centre_interet3 | centre_interet4 |
+----------------+----------------+----------------+----------------+
| premier        | deuxième       | troisième      | quatrième      |
| chevaux        | animaux        |                |                |
| musique        | pêche          | livres         | films          |
| peinture       |                |                |                |
| chevaux        | animaux        |                |                |
| musique        | sports         | livres         | navigation     |
| voyages        | musique        |                |                |
| chevaux        | animaux        |                |                |
| musique        | sport          | livres         | tricot         |
| animaux        | écriture       | voyages        |                |
| chiens         | randonnée      |                |                |
| films          | sport          |                |                |
+----------------+----------------+----------------+----------------+
```

Mais au moins, nous pouvons écrire quatre instructions SELECT séparées pour récupérer toutes les valeurs :

```
SELECT centre_interet1 FROM mes_contacts;
SELECT centre_interet2 FROM mes_contacts;
SELECT centre_interet3 FROM mes_contacts;
SELECT centre_interet4 FROM mes_contacts;
```

Maintenant, tout ce qu'il nous manque, c'est un moyen de prendre ces instructions SELECT pour en fourrer le contenu directement dans nos nouvelles tables. Il n'y a pas qu'un moyen de faire cela ; il en existe au moins trois !

À faire à la maison

Prenez l'instruction SELECT de la colonne profession que vous avez écrit page 345 :

```
SELECT profession FROM mes_contacts GROUP BY profession
ORDER BY profession;
```

Nous allons vous montrer sur la page suivante *TROIS FAÇONS* de profiter de ces instructions SELECT pour prépeupler vos nouvelles tables de centres d'intérêt.

Amusez-vous un peu avec SELECT, INSERT et CREATE pour voir ce que vous pouvez trouver. Puis regardez page suivante pour voir ces trois méthodes.

L'important ici n'est pas de trouver les réponses, mais de réfléchir aux possibilités.

Plusieurs chemins mènent à Rome

Même si pouvoir faire la même chose de trois ou plus de façons peut sembler amusant à des clowns un peu siphonnés, ce peut être déroutant pour le reste d'entre nous.

Mais c'est utile. Si vous connaissez trois façons de faire quelque chose, vous pouvez choisir celle qui correspond le mieux à vos besoins. Et quand vos données vont s'élargir, vous remarquerez que votre SGBDR exécute certaines requêtes plus rapidement. Quand vos tables deviennent très grandes, vous devez optimiser vos requêtes et savoir que vous pouvez réaliser la même tâche de plusieurs façons peut vous aider à le faire.

Dans les pages qui viennent vous trouverez les trois façons de créer et peupler cette table avec des valeurs distinctes et triées dans l'ordre alphabétique.

profession
id_prof
profession

CREATE, SELECT et INSERT (presque) en même temps

1. CREATion de TABLE, puis INSERTion avec le SELECT

Vous savez comment faire cela ! D'abord vous créez la table `profession` avec `CREATE`, puis vous peuplez les colonnes avec les valeurs de votre `SELECT` de la page 345.

Créez la table profession avec une colonne clé primaire et une colonne VARCHAR pour stocker les professions.

```
CREATE TABLE profession
(
  id_prof INT(11) NOT NULL AUTO_INCREMENT PRIMARY KEY,
  profession varchar(40)
);

INSERT INTO profession (profession)
  SELECT profession FROM mes_contacts
  GROUP BY profession
  ORDER BY profession;
```

Complétez maintenant la colonne profession de la table profession avec les valeurs de votre SELECT.

2. CREATion de TABLE avec SELECTion, puis ALTERation pour ajouter une clé primaire

Deuxième méthode : créer la table `profession` en utilisant les données d'un `SELECT` qui récupère les valeurs de la colonne profession de la table `mes_contacts`, puis `ALTERer` la table avec `ADD` pour ajouter le champ de la clé primaire.

```
CREATE TABLE profession AS
  SELECT profession FROM mes_contacts
  GROUP BY profession
  ORDER BY profession;

ALTER TABLE profession
ADD COLUMN id_prof INT NOT NULL AUTO_INCREMENT FIRST,
ADD PRIMARY KEY (id_prof);
```

Crée la table profession avec une colonne, remplie par les valeurs du SELECT...

... puis ALTERe la table pour ajouter le champ de la clé primaire

CREATE, SELECT et INSERT en même temps

3. CREATion de TABLE avec une clé primaire et avec SELECT en une fois

C'est la méthode en une étape : créer la table `profession` avec une colonne clé primaire et une colonne VARCHAR pour conserver les valeurs des professions et la remplir en même temps avec les valeurs du `SELECT`. SQL auto-incrémente, votre SGBDR sait donc que la colonne id_prof doit être complétée automatiquement et cela ne laisse plus qu'une colonne, c'est donc là que vont les données.

Crée la table profession avec les deux colonnes clé primaire et profession et remplit la colonne profession avec les valeurs du SELECT.

```
CREATE TABLE profession
(
  id_prof INT(11) NOT NULL AUTO_INCREMENT PRIMARY KEY,
  profession varchar(40)
) AS
  SELECT profession FROM mes_contacts
  GROUP BY profession
  ORDER BY profession;
```

Je n'ai jamais vu AS avant. On dirait qu'on l'utilise pour référencer les résultats d'une requête et les insérer dans la nouvelle table.

Oui. Le mot-clé AS fait exactement ce qu'il semble faire.

Tout cela fait partie des alias, auxquels nous arrivons justement !

Que change un AS ?

AS peuple une nouvelle table avec les résultats d'un SELECT. Donc quand nous avons utilisé AS dans le deuxième et le troisième exemple, nous avons indiqué au logiciel de prendre toutes les valeurs résultant du SELECT effectué sur la table mes_contacts et de les mettre dans la nouvelle table profession tout juste créée.

Si nous n'avions pas spécifié que la nouvelle table avait deux colonnes avec de nouveaux noms, AS aurait créé une seule colonne, remplie avec le même nom et le même type de données que la colonne renvoyée par le SELECT.

Si nous n'avions pas donné deux colonnes à la nouvelle table, AS aurait créé une colonne avec le même nom et le même type de données que la colonne renvoyée par le SELECT.

Nous créons une colonne VARCHAR dans notre nouvelle table et l'appelons profession.

```
CREATE TABLE profession
(
  id_prof INT(11) NOT NULL AUTO_INCREMENT PRIMARY KEY,
  profession varchar(20)
) AS
  SELECT profession FROM mes_contacts
  GROUP BY profession
  ORDER BY profession;
```

Ce petit mot-clé fait de grandes choses. Il canalise tous les résultats du SELECT vers la nouvelle table.

Ceux-ci font tous référence à la colonne profession de mes_contacts parce qu'ils font partie du SELECT.

Comme nous avons créé la table profession avec une clé primaire auto-incrementée, nous n'avons plus qu'à ajouter les valeurs à la seconde colonne de cette table, que nous avons appelée profession.

Je suis perdue. "profession" apparaît cinq fois dans cette seule requête. Le logiciel SQL sait peut-être quelle profession est quoi, mais *moi*, comment puis-je faire la différence ?

SQL vous permet d'assigner un alias à un nom de colonne pour ne pas vous y perdre.

C'est l'une des raisons pour lesquelles SQL vous permet de donner temporairement de nouveaux noms à vos tables et vos colonnes, appelés des **alias**.

Alias de colonnes

Créer un alias est extrêmement facile. Nous allons le placer juste après la première utilisation du nom de colonne dans notre requête avec un autre `AS` pour dire à notre logiciel que cela renvoie à la colonne `profession` de `mes_contacts` sous un nouveau nom, qui facilite la compréhension de ce qui se passe.

Nous allons appeler **`mc_prof`** les valeurs de professions que nous sélectionnons de la table `mes_contacts` (`mc` est l'abréviation de `mes_contacts`).

```
CREATE TABLE profession
(
  id_prof INT(11) NOT NULL AUTO_INCREMENT PRIMARY KEY,
  profession varchar(20)
) AS
  SELECT profession AS mc_prof FROM mes_contacts
  GROUP BY mc_prof
  ORDER BY mc_prof;
```

Placez votre alias juste après la première occurence du nom de colonne d'origine dans la requête pour indiquer à votre logiciel que l'alias s'y réfère à partir de maintenant.

Cette requête fait exactement la même chose que la première, mais grâce à l'alias, elle est plus facile à comprendre.

Il y a une petite différence entre les deux requêtes. Elles renvoient toutes les deux les résultats sous forme de tables. ***L'alias change le nom de la colonne* dans les résultats** mais il ***ne modifie en aucune façon la colonne d'origine***. Un alias est **temporaire**.

Mais puisque nous sommes passé outre les résultats en spécifiant que notre nouvelle table a deux colonnes (la colonne de la clé primaire et notre colonne profession), notre nouvelle table a toujours une colonne appelée `profession`, et non pas `mc_prof`.

profession
programmeur
professeur
avocat

Les résultats de la requête d'origine avec le nom de colonne d'origine.

Les résultats de la requête utilisant l'alias. Le nom de la colonne est le même que celui de l'alias.

mc_prof
programmeur
professeur
avocat

Des alias de tables, qui aurait besoin de ça ?

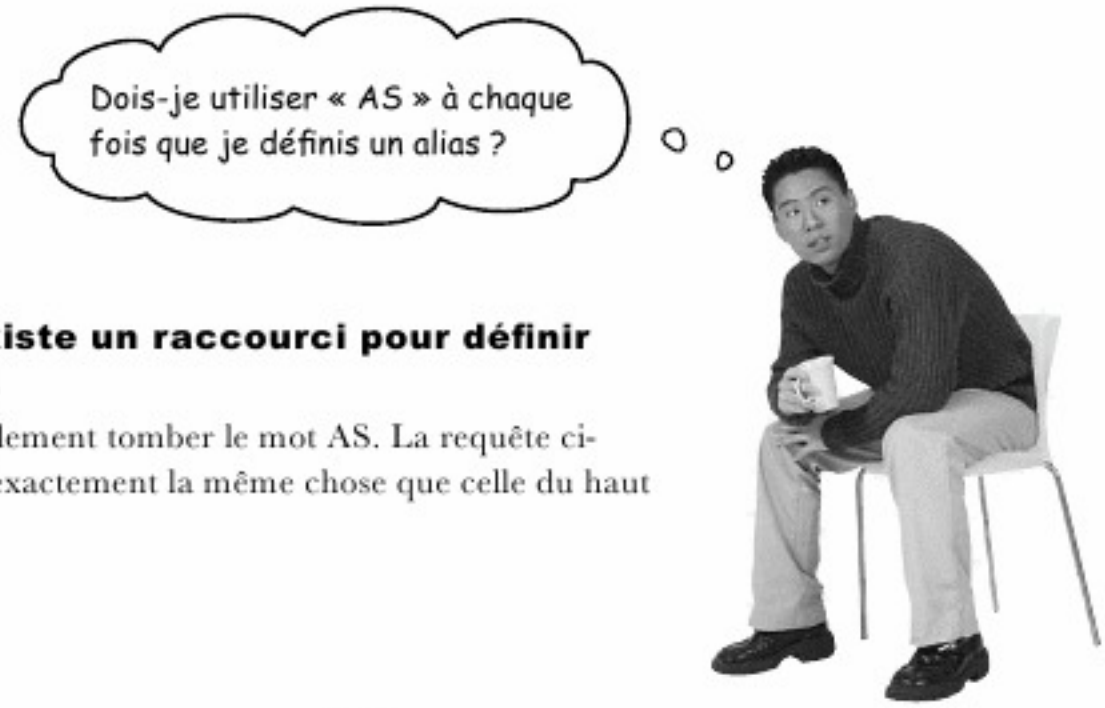
Vous ! Nous sommes sur le point de plonger tête la première dans le monde des jointures, où nous sélectionnons des données de plus d'une table. Et sans les alias, vous allez vite vous fatiguer de taper ces noms de table sans arrêt.

Vous créez des alias de tables de la même façon que vous créez des alias de colonnes. Placez l'alias de table après la première occurencence du nom de la table dans la requête, avec un autre `AS` pour indiquer à votre logiciel de se référer à la table `mes_contacts` sous le pseudo de **mc** à partir de maintenant.

On appelle aussi les alias de table des noms de corrélation

```
SELECT profession AS mc_prof
FROM mes_contacts AS mc
GROUP BY mc_prof
ORDER BY mc_prof;
```

Créez votre alias de table exactement de la même façon que vous créez vos alias de colonnes.



Non, il existe un raccourci pour définir vos alias.

Laissez simplement tomber le mot AS. La requête ci-dessous fait exactement la même chose que celle du haut de la page.

Il n'y a pas de différence entre ce que font ces deux requêtes.

```
SELECT profession mc_prof
FROM mes_contacts mc
GROUP BY mc_prof
ORDER BY mc_prof;
```

Nous avons enlevé l'AS. Cela fonctionne tant que l'alias suit directement le nom de la colonne ou de la table dont il est l'alias.

Tout ce que vous avez toujours voulu savoir sur les jointures internes

Si vous avez déjà entendu quelqu'un parler de SQL, vous avez probalement entendu le mot « jointure » ici ou là. Elles ne sont pas aussi compliquées que vous pouvez le penser. Nous allons vous emmener en balade parmi elles, vous montrer comment elles fonctionnent et vous donner de nombreuses occasions de comprendre quand vous devez les utiliser et laquelle choisir.

Mais, pour commencer, voyons le type de jointure le plus simple (qui n'est pas du tout une vraie jointure).

Elle a plusieurs noms. Dans ce livre, nous l'appellerons une **jointure cartésienne**, mais on l'appelle aussi produit cartésien, produit croisé, jointure croisée et, assez bizarrement, « non-jointure ».

Supposons que vous ayez une table de noms d'enfants et une autre table avec les jouets qui leur appartiennent. À vous de trouver quels jouets vous pouvez acheter à chaque enfant.

jouets

id_jouet	jouet
1	cerceau
2	maquette d'avion
3	petits soldats
4	harmonica
5	jeu de cartes

enfants

id_enfant	enfant
1	David
2	Adrien
3	Vincent
4	Colin

Jointure cartésienne

La requête ci-dessous nous donne les résultats cartésiens quand nous interrogeons les deux tables en même temps : la colonne jouet de jouets et la colonne enfant d'enfants.

```
SELECT j.jouet, e.enfant
FROM jouets AS j
  CROSS JOIN
  enfants AS e;
```

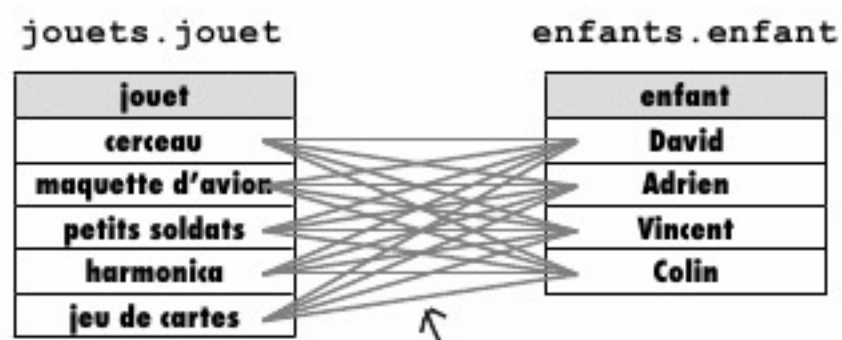
Vous souvenez-vous de nos notations abrégées du dernier chapitre ? Le nom avant le point est la table et le nom d'après est le nom d'une colonne de cette table. Seulement cette fois, nous utilisons les alias des tables plutôt que leurs noms entiers.

Ici aussi nous utilisons les alias des tables.

Cette ligne dit : SELECTionne la colonne appelée 'enfant' de la table enfants et la colonne appelée 'jouet' de la table jouets. Et le reste de la requête joint ces deux colonnes dans une nouvelle table de résultats.

La jointure cartésienne prend chaque valeur de la première table et la groupe avec chaque valeur de la deuxième table.

La JOINTURE CROISÉE renvoie toutes les lignes d'une table croisées avec toutes les lignes de la seconde table.



Ces lignes montrent les résultats de la jointure. Chaque jouet est associé avec chaque enfant. Il n'y a aucun duplicat.

Cette jointure nous donne 20 résultats. Cela fait 5 jouets x 4 enfants pour représenter toutes les combinaisons possibles.

Ce n'est que parce que jouets.jouet avait plus de résultats que les groupes se font par jouet. Si nous avions 5 résultats pour enfant et 4 pour jouet, ce serait groupé par nom d'enfant. Mais rappelez-vous, l'ordre des résultats n'a aucune signification avec cette requête.

jouet	enfant
cerceau	David
cerceau	Adrien
cerceau	Vincent
cerceau	Colin
maquette d'avion	David
maquette d'avion	Adrien
maquette d'avion	Vincent
maquette d'avion	Colin
petits soldats	David
petits soldats	Adrien

Q: Comment pourrais-je avoir besoin de cela ?

R: Il est important de la connaître parce que si vous faites n'importe quoi avec les jointures, vous risquez d'obtenir des résultats cartésiens par accident. Ceci vous aidera à trouver comment réparer votre jointure. Cela peut vraiment arriver. De plus, on utilise parfois des jointures croisées pour tester la vitesse et la configuration de votre SGBDR. Il est plus facile de détecter et de comparer le temps que prend une requête si vous en prenez une lente.

Q: Disons que j'utilise cette requête à la place :
`SELECT * FROM jouets CROSS JOIN enfants;`
que se passe-t-il si j'utilise SELECT * ?

R: Vous devriez essayer vous-même. Mais vous obtiendriez toujours 20 lignes ; elles inclueraient simplement les 4 colonnes.

Une JOINTURE INTERNE est une JOINTURE CROISÉE dont certaines lignes de résultat ont été enlevées par une condition de la requête.

Q: Et si je fais une jointure croisée avec deux très grosses tables ?

R: Vous obtiendrez un nombre de résultats ***énorme***. Il vaut mieux ne pas faire de jointure croisée avec de grandes tables, vous courez le risque de noyer votre machine sous la quantité de données à renvoyer !

Q: Existe-t-il une autre syntaxe pour cette requête ?

R: Gagné. Vous pouvez enlever les mots CROSS JOIN et utiliser une simple virgule à la place, comme ceci :

```
SELECT jouets.jouet, enfants.enfant
FROM jouets, enfants;
```

Q: J'ai déjà entendu les termes « jointure interne » et « jointure externe » auparavant. Cette jointure cartésienne est-elle la même chose ?

R: Une jointure cartésienne est un type de jointure interne. À la base, une jointure interne est tout simplement une jointure cartésienne où certaines des lignes de résultats sont retirées par une condition de la requête. Nous allons nous attarder sur les jointures internes dans les prochaines pages, alors gardez tout cela à l'esprit !

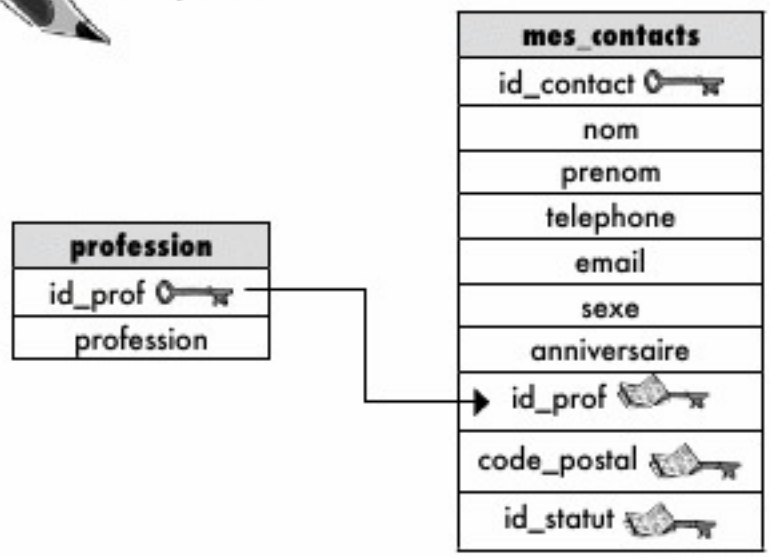
GYM DU CERVEAU

Quel serait le résultat de cette requête à votre avis ?

```
SELECT e1.enfant, e2.enfant
FROM enfants AS e1 CROSS JOIN enfants AS e2;
```

Essayez-la.

À vos crayons !



Voici deux tables de la structure de la base de données : profession et mes_contacts. Regardez la requête et écrivez en face ce que fait chacune des ligne selon vous.

```
SELECT mc.nom,                    .....
mc.prenom,                        .....
p.profession                      .....
FROM mes_contacts AS mc           .....
  INNER JOIN                      .....
  profession AS p                 .....
ON mc.id_prof = p.id_prof;        .....
                                  .....
```

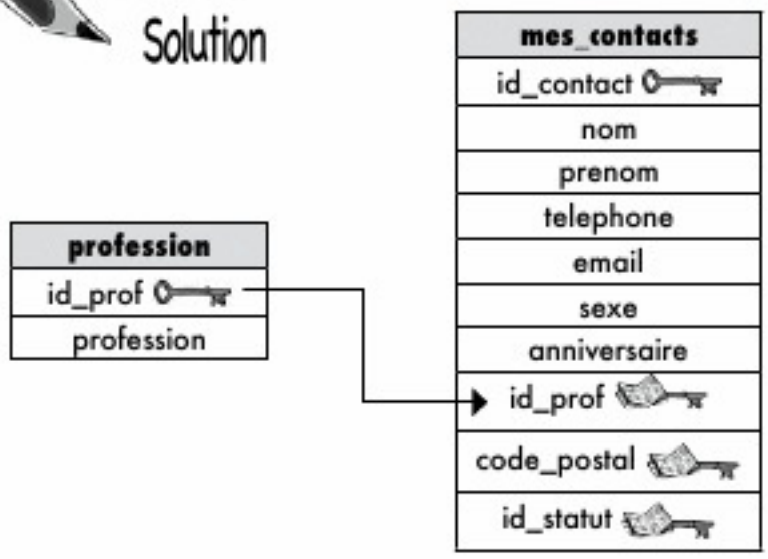
Supposez que les données des trois pense-bêtes ci-dessous se trouvent dans les tables.
Dessinez la table des résultats à partir de ces fiches.

Janine Cuvelier
célibataire
3-4-1978
Lyon, 69
Artiste
femme
jcuvelier@distribon.net
voile, randonnée, cuisine
04-98-76-54-32

Bénédicte Bérêt
mariée
1-9-1970
Strasbourg, 67
Chef
femme
bene@pizzaprestissima.com
films, lecture, cuisine
03-88-08-80-00

Paul Singer
marié
10-12-1980
Paris, 75
Professeur
homme
ps@bardescaraibes.com
chiens, spéléologie
01-88-88-00-66

À vos crayons ! Solution



Voici deux tables de la structure de la base de données : profession et mes_contacts. Regardez la requête et écrivez en face ce que fait chacune des ligne selon vous.

```
SELECT mc.nom,
```
SELECTionne la colonne nom de la table mes_contacts (alias mc)

```
mc.prenom,
```
et la colonne prenom de la table mes_contacts

```
p.profession
```
et la colonne profession de la table profession (alias p)

```
FROM mes_contacts AS mc
```
dans la table mes_contacts (alias mc) et

```
  INNER JOIN
```
utilise une JOINTURE INTERNE pour joindre les résultats du SELECT avec

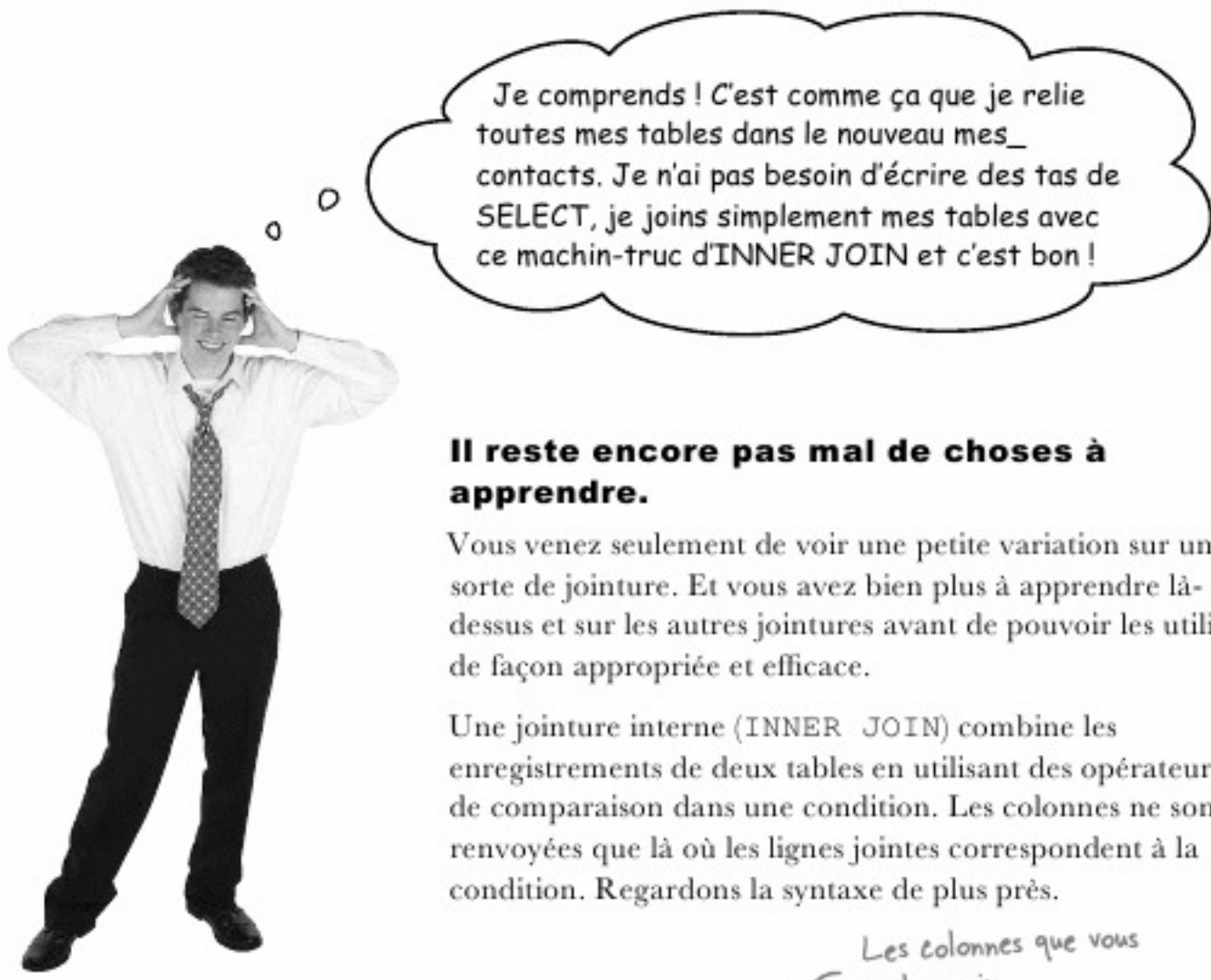
```
  profession AS p
```
la table profession (alias p)

```
ON mc.id_prof = p.id_prof;
```
là où l'id_prof de mes_contacts correspond au champ id_prof de la table profession

Supposez que les données des trois pense-bêtes ci-dessous se trouvent dans les tables.
Dessinez la table des résultats à partir de ces fiches.

nom	prenom	profession
Cuvelier	Janine	artiste
Singer	Paul	professeur
Bérêt	Bénédicte	chef

Libérez votre jointure interne avec INNER JOIN



Il reste encore pas mal de choses à apprendre.

Vous venez seulement de voir une petite variation sur une sorte de jointure. Et vous avez bien plus à apprendre là-dessus et sur les autres jointures avant de pouvoir les utiliser de façon appropriée et efficace.

Une jointure interne (`INNER JOIN`) combine les enregistrements de deux tables en utilisant des opérateurs de comparaison dans une condition. Les colonnes ne sont renvoyées que là où les lignes jointes correspondent à la condition. Regardons la syntaxe de plus près.

Les colonnes que vous voulez voir.

```
SELECT descolonnes
FROM table1
  INNER JOIN
  table2
ON unecondition;
```

Nous avons laissé les alias de côté pour simplifier les choses.

Le mot-clé WHERE fonctionnera aussi ici.

Cette condition peut utiliser n'importe quel opérateur de comparaison.

Une JOINTURE INTERNE combine les enregistrements de deux tables en utilisant des opérateurs de comparaison dans une condition.

La jointure interne au travail : l'équi-jointure

Prenez ces tables. Chaque enfant a un seul jouet. Nous avons une relation un-à-un et `id_jouet` est une clé étrangère.

enfants

id_enfant	enfant	id_jouet
1	David	3
2	Adrien	5
3	Vincent	2
4	Colin	1

jouets

id_jouet	jouet
1	cerceau
2	maquette d'avion
3	petits soldats
4	harmonica
5	jeu de cartes

Nous voulons simplement trouver quel jouet a chaque enfant. Nous pouvons utiliser notre jointure interne avec l'opérateur = pour faire correspondre la clé étrangère d'enfants à la clé primaire de jouets et voir à quel jouet cela nous mène.

```
SELECT enfants.enfant, jouets.jouet
FROM enfants
  INNER JOIN
  jouets
ON enfants.id_jouet = jouets.id_jouet;
```

La jointure interne ÉQUI-JOINTURE teste une égalité

enfants

id_enfant	enfant	id_jouet
1	David	3
2	Adrien	5
3	Vincent	2
4	Colin	1

jouets

id_jouet	jouet
1	cerceau
2	maquette d'avion
3	petits soldats
4	harmonica
5	jeu de cartes

Notre table de résultats. On peut ajouter un ORDER BY enfants.enfant si l'on veut.

enfant	jouet
Colin	cerceau
Vincent	maquette d'avion
David	petits soldats
Adrien	jeu de cartes

À vos crayons !

Écrivez les requêtes d'équi-jointure pour la base de données liste_greg ci-dessous.

Une requête qui renvoie les adresses email et les professions de chaque personne de mes_contacts.

.....

.....

.....

Une requête qui renvoie le prénom, nom et statut de chaque personne de mes_contacts.

.....

.....

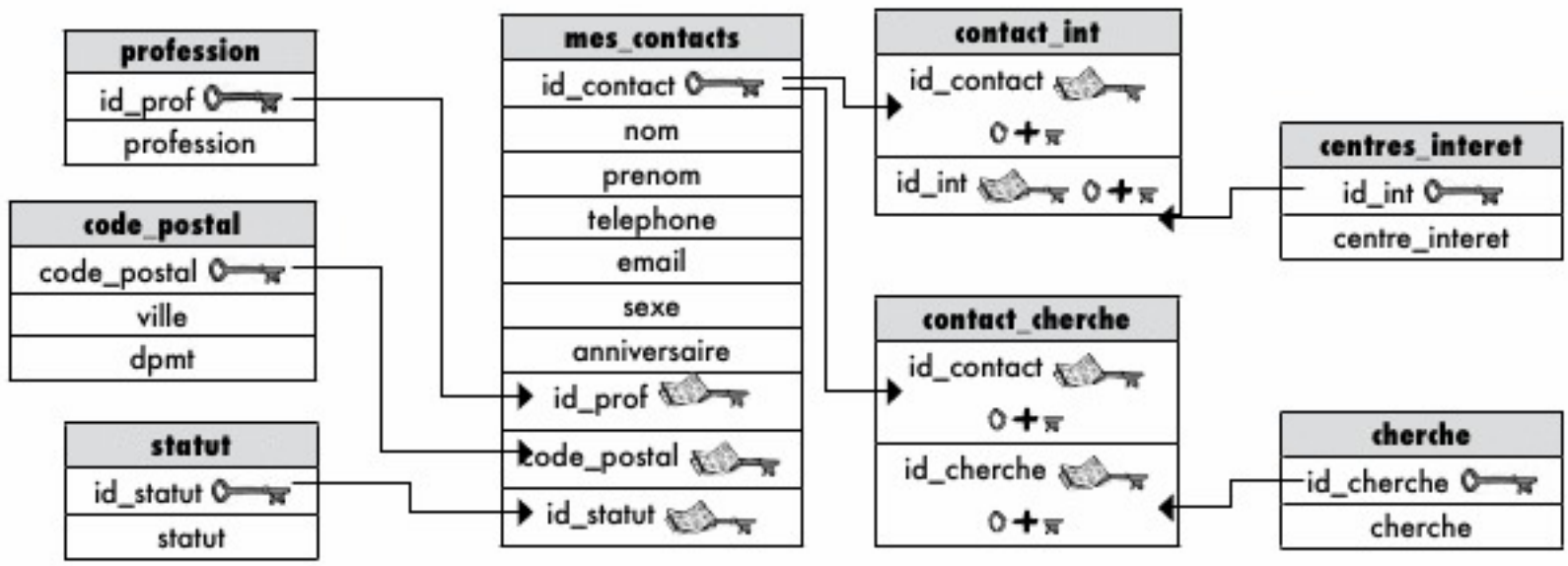
.....

Une requête qui renvoie le prénom, le nom et le département de chaque personne de mes_contacts.

.....

.....

.....



À vos crayons ! Solution

Écrivez les requêtes d'équi-jointure pour la base de données liste_greg ci-dessous.

Une requête qui renvoie les adresses email et les professions de chaque personne de mes_contacts.

```
SELECT me.email, p.profession FROM mes_contacts me
INNER JOIN profession p ON me.id_prof = p.id_prof;
```

La clé étrangère id_prof relie à l'id_prof de la table profession.

Une requête qui renvoie le prénom, nom et statut de chaque personne de mes_contacts.

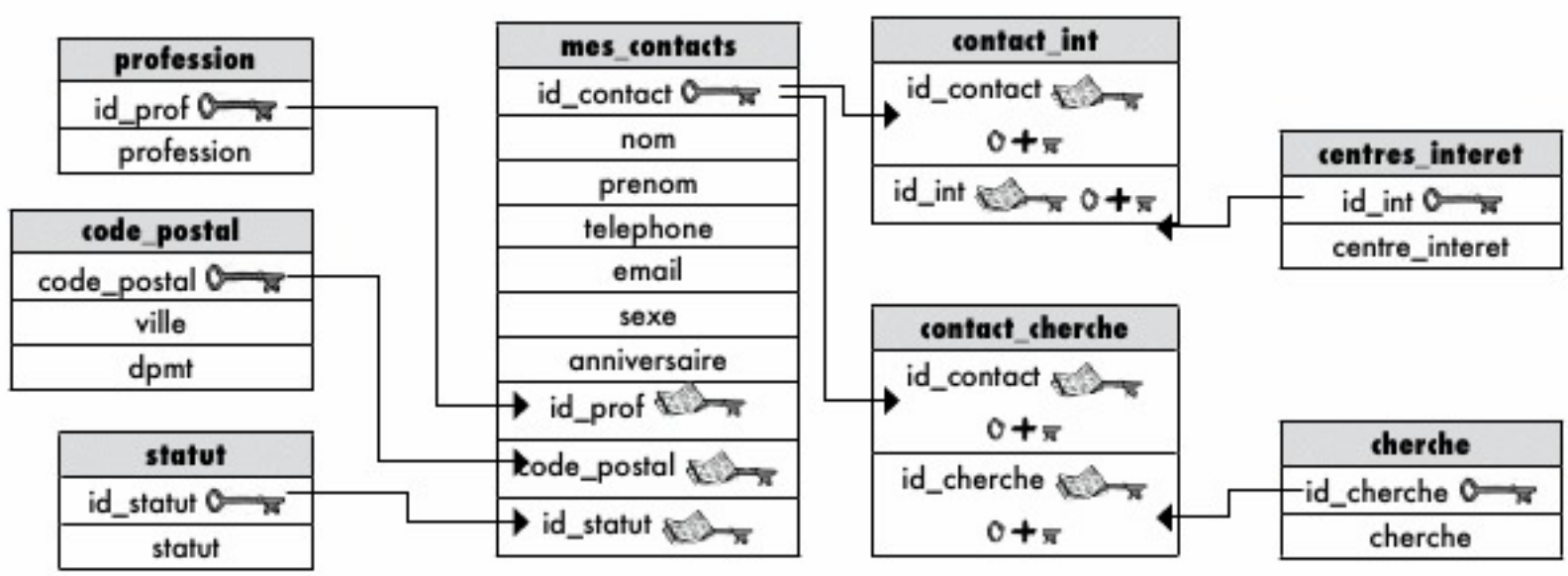
```
SELECT me.prenom, me.nom, s.statut FROM mes_contacts me
INNER JOIN statut s ON me.id_statut = s.id_statut;
```

La clé étrangère id_statut relie à l'id_statut de la table statut.

Une requête qui renvoie le prénom, le nom et le département de chaque personne de mes_contacts.

```
SELECT me.prenom, me.nom, cp.dpmt FROM mes_contacts me
INNER JOIN code_postal cp ON me.code_postal = cp.code_postal;
```

Cette fois nous utilisons code_postal comme clé pour relier les deux tables.



La jointure interne au travail : la non équi-jointure

La **non équi-jointure** renvoie toutes les lignes qui ne sont pas égales. Reprenons les deux tables enfants et jouets. En utilisant la non équi-jointure, nous pouvons voir exactement quels jouets chaque enfant n'a ***pas*** (ce qui peut servir pour leurs anniversaires).

```
SELECT enfants.enfant, jouets.jouet
FROM enfants
  INNER JOIN
  jouets
ON enfants.id_jouet <> jouets.id_jouet
ORDER BY enfants.enfant;
```

Différent de. C'est la partie « non-equi » de la jointure.

Trier les résultats les rendra plus faciles à lire.

enfants

id_enfant	enfant	id_jouet
1	David	3
2	Adrien	5
3	Vincent	2
4	Colin	1

jouets

id_jouet	jouet
1	cerceau
2	maquette d'avion
3	petits soldats
4	harmonica
5	jeu de cartes

enfant	jouet
Adrien	petits soldats
Adrien	harmonica
Adrien	cerceau
Adrien	maquette d'avion
Colin	maquette d'avion
Colin	petits soldats
Colin	harmonica
Colin	jeu de cartes
David	cerceau
David	maquette d'avion
David	harmonica
David	jeu de cartes
Vincent	cerceau
Vincent	petits soldats
Vincent	harmonica
Vincent	jeu de cartes

Voici les quatre jouets qu'Adrien n'a pas encore.

La jointure interne NON ÉQUI-JOINTURE teste une inégalité.

La dernière jointure interne : la jointure naturelle

Il ne reste plus qu'un type de jointure interne et elle s'appelle **jointure naturelle**. Les jointures naturelles ne fonctionnent que si la **colonne par laquelle vous joignez a le *même nom dans les deux tables***. Considérons de nouveau ces deux tables.

Même nom de colonne

enfants

id_enfant	enfant	id_jouet
1	David	3
2	Adrien	5
3	Vincent	2
4	Colin	1

jouets

id_jouet	jouet
1	cerceau
2	maquette d'avion
3	petits soldats
4	harmonica
5	jeu de cartes

Tout comme précédemment, nous voulons savoir quel jouet a chaque enfant. Notre jointure naturelle va reconnaître le même nom de colonne dans chaque table et renvoyer les lignes correspondantes.

```
SELECT enfants.enfant, jouets.jouet
FROM enfants
  NATURAL JOIN
  jouets;
```

enfants

id_enfant	enfant	id_jouet
1	David	3
2	Adrien	5
3	Vincent	2
4	Colin	1

jouets

id_jouet	jouet
1	cerceau
2	maquette d'avion
3	petits soldats
4	harmonica
5	jeu de cartes

Nous obtenons exactement le même ensemble de résultats qu'avec notre première jointure interne, l'équi-jointure.

enfant	jouet
Colin	cerceau
Vincent	maquette d'avion
David	petits soldats
Adrien	jeu de cartes

La jointure interne JOINTURE NATURELLE identifie les noms de colonnes correspondants.

À vos crayons !

Écrivez les requêtes pour la base de données liste_greg ci-dessous avec des jointures naturelles ou des non équi-jointures :

Une requête qui renvoie les adresses email et les professions de chaque personnes de mes_contacts.

.....

.....

.....

Une requête qui renvoie le prénom, le nom de chaque personne de mes_contacts et les statuts qu'elle n'a pas.

.....

.....

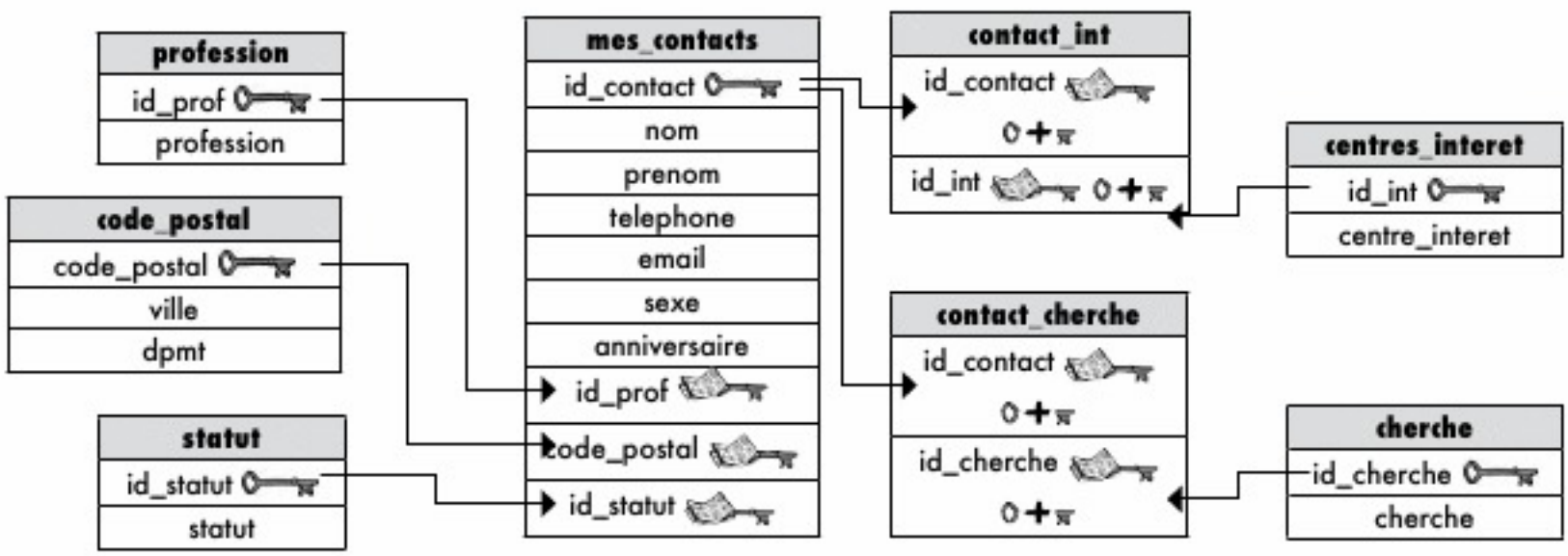
.....

Une requête qui renvoie le prénom, le nom et le département de chaque personne de mes_contacts.

.....

.....

.....



À vos crayons ! Solution

Écrivez les requêtes pour la base de données liste_greg ci-dessous avec des jointures naturelles ou des non équi-jointures :

Une requête qui renvoie les adresses email et les professions de chaque personnes de mes_contacts.

```
SELECT me.email, p.profession FROM mes_contacts me
NATURAL JOIN profession p;
```

Une requête qui renvoie le prénom, le nom de chaque personne de mes_contacts et les statuts qu'elle n'a pas.

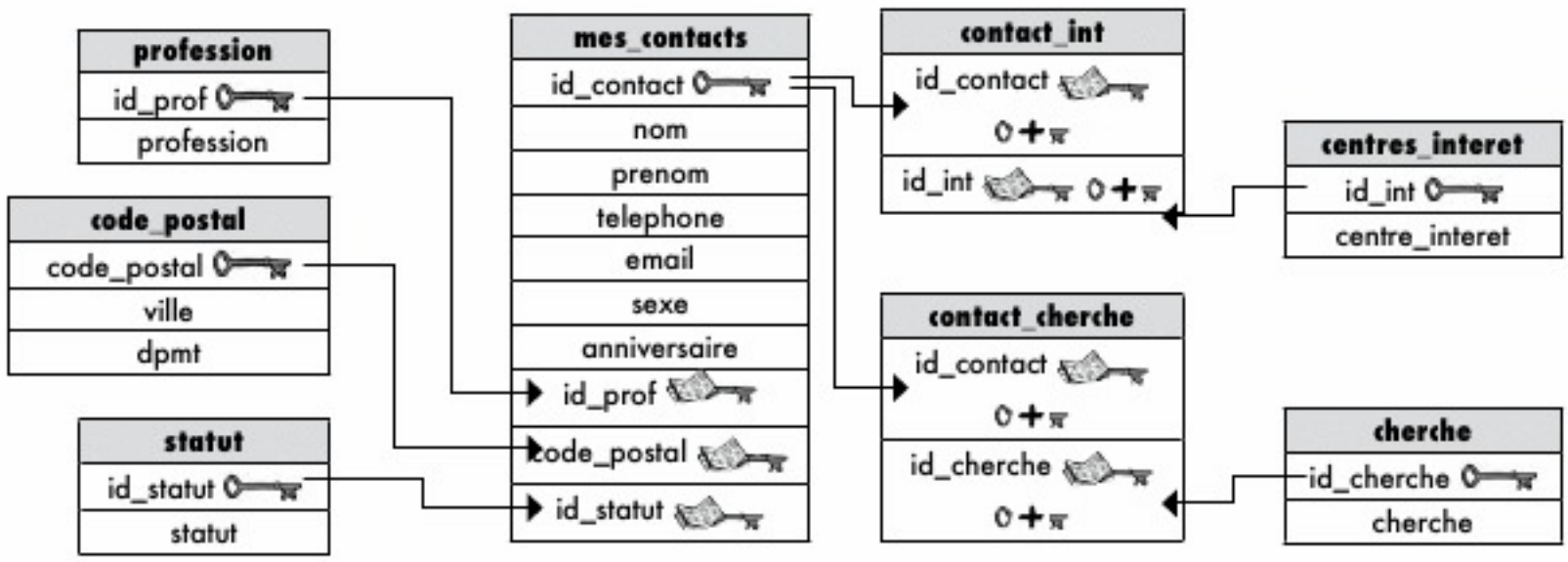
```
SELECT me.prenom, me.nom, s.statut FROM mes_contacts me
INNER JOIN statut s ON me.id_statut <> s.id_statut;
```

Vous obtiendrez de nombreuses lignes pour chaque personne, avec les statuts qui ne sont pas reliés par id_statut.

Une requête qui renvoie le prénom, le nom et le département de chaque personne de mes_contacts.

```
SELECT me.prenom, me.nom, cp.dpmt FROM mes_contacts me
NATURAL JOIN code_postal cp;
```

Nous n'avons pas besoin de la partie ON dans la première ou la troisième requête parce que notre clé étrangère y a le même nom que la clé primaire.



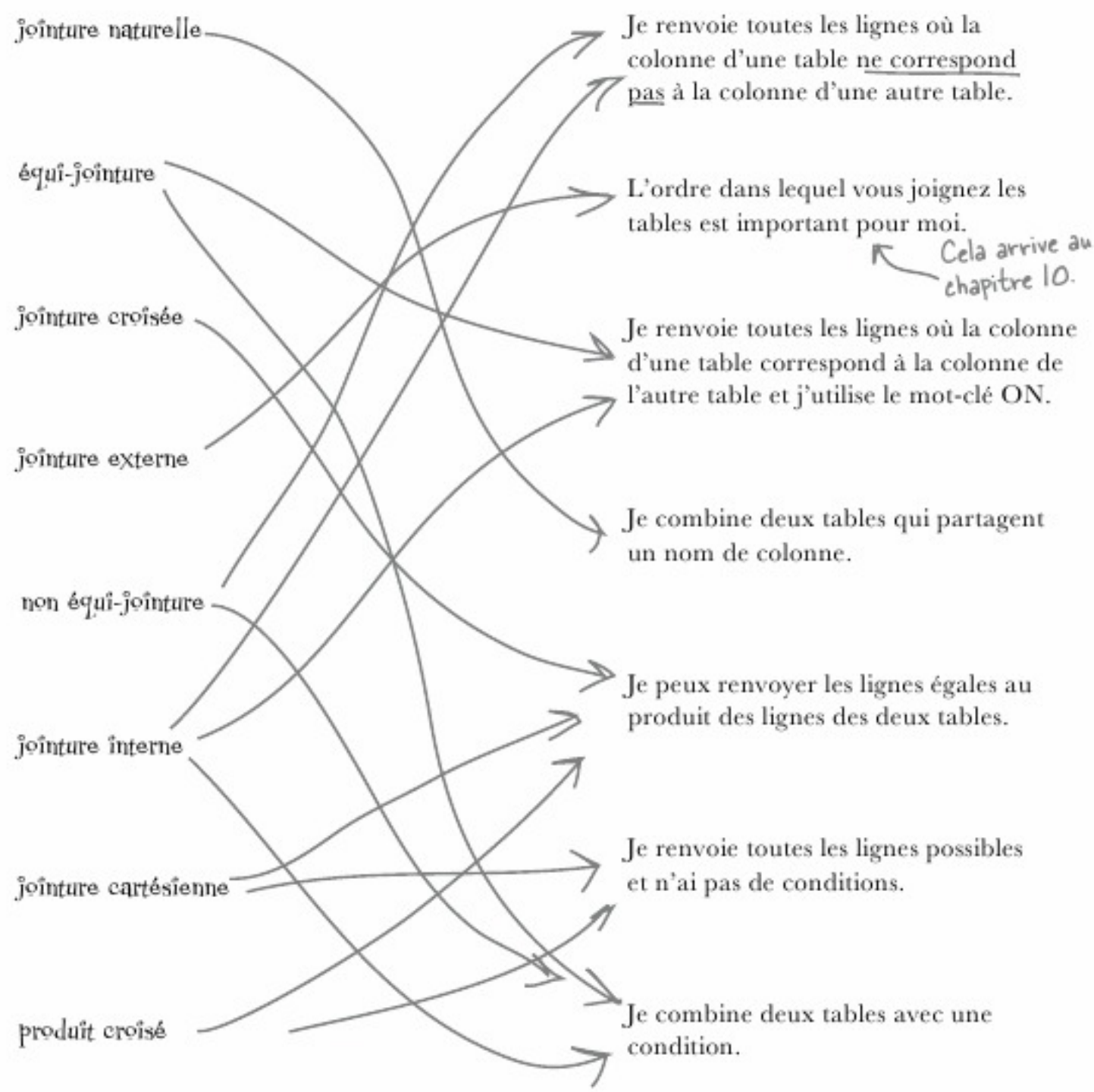
qui fait quoi ?

Reliez chaque jointure à la description de ce qu'elle fait. Plus d'une jointure peuvent correspondre à une description.

jointure naturelle	Je renvoie toutes les lignes où la colonne d'une table ne correspond pas à la colonne d'une autre table.
équi-jointure	L'ordre dans lequel vous joignez les tables est important pour moi.
jointure croisée	Je renvoie toutes les lignes où la colonne d'une table correspond à la colonne de l'autre table et j'utilise le mot-clé ON.
jointure externe	Je combine deux tables qui partagent un nom de colonne.
non équi-jointure	Je peux renvoyer les lignes égales au produit des lignes des deux tables.
jointure interne	Je renvoie toutes les lignes possibles et n'ai pas de conditions.
jointure cartésienne	Je combine deux tables avec une condition.
produit croisé	

QUI FAIT QUOI ?

Reliez chaque jointure à la description de ce qu'elle fait. Plus d'une jointure peuvent correspondre à une description.



Utilisez le diagramme de la base de données liste_greg ci-dessous pour écrire des requêtes SQL pour obtenir les informations voulues.

Écrivez deux requêtes, chacune avec une jointure différente, pour obtenir les enregistrements correspondants de mes_contacts et contact_int.

.....

.....

.....

Écrivez une requête pour renvoyer toutes les combinaisons possibles des lignes de contact_cherche et cherche.

.....

.....

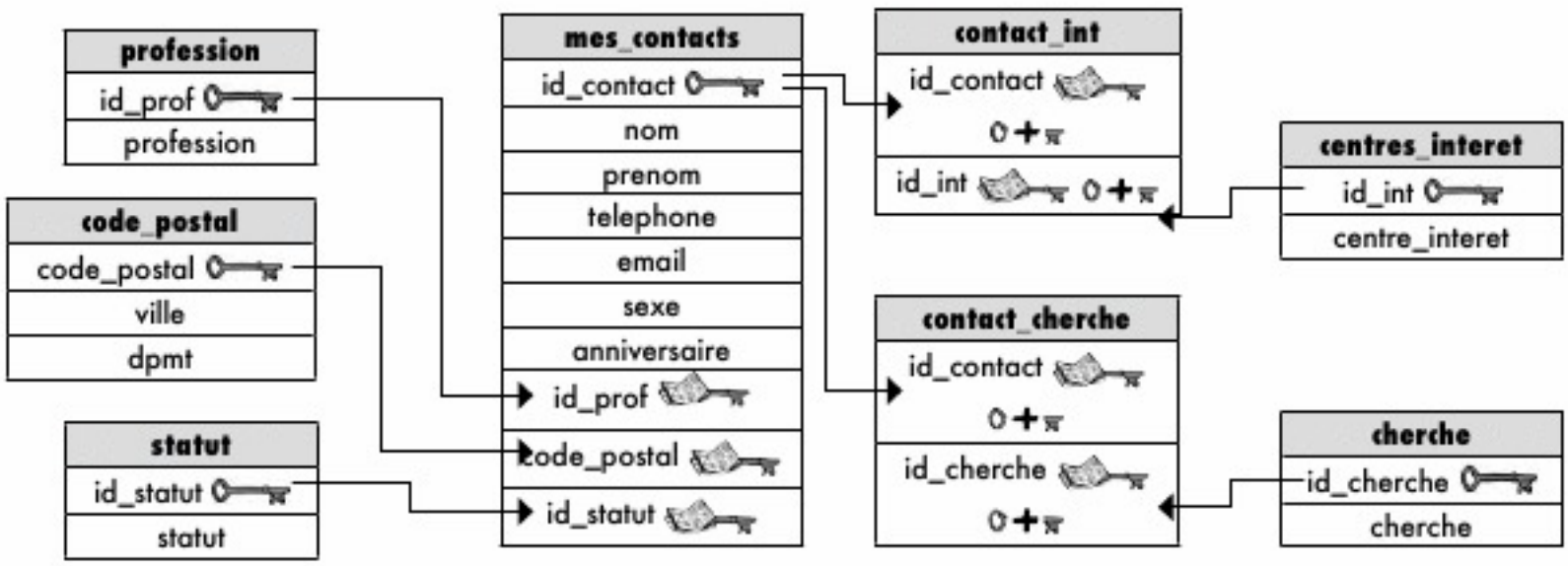
.....

Listez les professions des gens de la table mes_contacts, mais sans duplicats et dans l'ordre alphabétique.

.....

.....

.....



Solutions des exercices

Utilisez le diagramme de la base de données liste_greg ci-dessous pour écrire des requêtes SQL pour obtenir les informations voulues.

Écrivez deux requêtes, chacune avec une jointure différente, pour obtenir les enregistrements correspondants de mes_contacts et contact_int.

```
SELECT mc.prenom, mc.nom, ci.id_int FROM mes_contacts mc
INNER JOIN contact_int ci ON mc.id_contact = ci.id_contact;

SELECT mc.prenom, mc.nom, ci.id_int FROM mes_contacts mc
NATURAL JOIN contact_int ci;
```

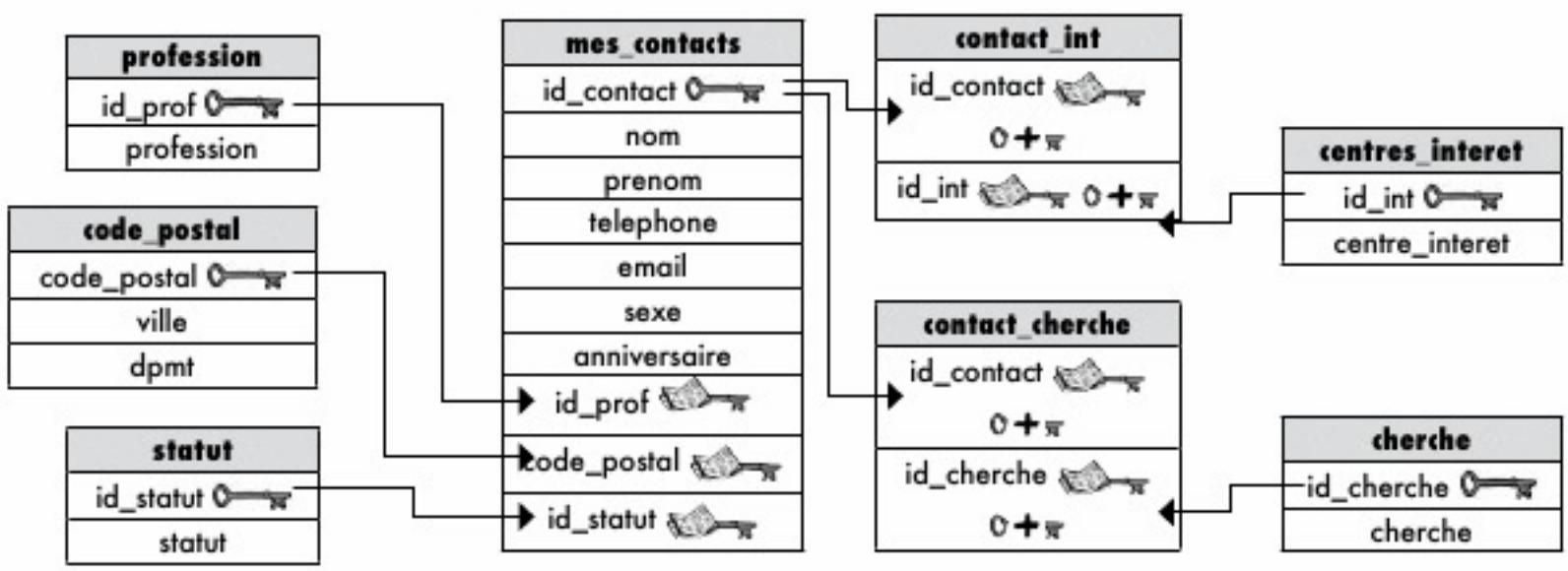
Écrivez une requête pour renvoyer toutes les combinaisons possibles des lignes de contact_cherche et cherche.

```
SELECT * FROM contact_cherche CROSS JOIN cherche;
SELECT * FROM contact_cherche, cherche;
```

Il y a deux façons de faire la même jointure croisée.

Listez les professions des gens de la table mes_contacts, mais sans duplicats et dans l'ordre alphabétique.

```
SELECT p.profession FROM mes_contacts mc
INNER JOIN profession p ON mc.id_prof = p.id_prof GROUP BY profession ORDER BY profession;
```



il n'y a pas de
Questions stupides

Q: Peut-on joindre plus de deux tables ?

R: On peut le faire et nous allons en parler un peu plus tard. Pour l'instant nous allons nous concentrer sur les concepts de jointures.

Q: Les jointures ne sont-elles pas censées être plus difficiles que cela ?

R: Une fois que vous commencez à entrer dans les jointures et les alias, les requêtes SQL ressemblent de plus en plus à une langue étrangère. De même, utiliser les raccourcis (comme remplacer les mots-clés INNER JOIN avec des virgules dans les requêtes, par exemple) peut rendre les choses encore plus déroutantes. Pour cette raison, ce livre préfère des requêtes SQL verbeuses plutôt que des raccourcis moins clairs.

Q: Cela signifie-t-il qu'il existe d'autres façons d'écrire des requêtes de jointures internes ?

R: Il en existe d'autres, oui. Mais si vous comprenez celles-ci, avec la syntaxe que nous présentons, comprendre la syntaxe des autres sera facile. Les concepts sont beaucoup plus importants que d'utiliser `WHERE` ou `ON` dans une jointure.

Q: J'ai remarqué que vous utilisez un ORDER BY dans une jointure. Cela signifie-t-il que tout le reste est jouable aussi ?

R: Oui. Ne vous gênez pas pour utiliser des clauses `GROUP BY`, `WHERE` et des fonctions comme `SUM` et `AVG`.

Des requêtes jointes ?

Greg commence à vraiment apprécier les jointures. Il commence à réaliser qu'avoir plusieurs tables est logique et qu'il n'est pas difficile de travailler avec elles si elles sont bien conçues. Il a même des projets pour élargir `liste_greg`.

Une requête à l'intérieur d'une autre requête ? Est-ce possible ?

Révélations sur les alias de tables et de colonnes

L'interview de la semaine : De quoi vous cachez-vous ?

Tête La Première : Bienvenue à Alias de Table et Alias de Colonne. Nous sommes heureux que vous puissiez être là tous les deux. Nous espérons que vous pouvez éclaircir certains points confus pour nous.

Alias de Table : Certainement, content d'être là. Et vous pouvez nous appeler **AT** et **AC** pendant cette interview, c'est plus court [rires].

TLP : Ha ha! En tout cas, c'est de circonstance. D'accord, **AC**, commençons par vous. Pourquoi tous ces mystères ? Essayez-vous de cacher quelque chose ?

Alias de Colonne : Absolument pas ! Au mieux, j'essaie de rendre les choses plus claires. Je pense que je parle en nos deux noms là, n'est-ce pas AT ?

AT : Tout à fait. Dans le cas d'AC, ce qu'il fait doit déjà être évident. Il prend des noms de colonnes longs et redondants et les rend plus faciles à suivre. Plus accessibles. Il vous donne aussi les tables de résultats avec des noms de colonnes utiles. Pour moi, c'est un peu différent.

TLP : Je dois admettre que je ne vous connais pas très bien, AT. Je vous ai vu à l'oeuvre, mais je ne suis toujours pas certain de ce que vous faites. Vous n'apparaissez pas du tout dans les résultats quand nous vous utilisons dans une requête.

AT : Oui, c'est vrai. Mais je pense que vous ne saisissez pas encore ma vocation supérieure.

TLP : Vocation supérieure ? Ça a l'air fascinant. Continuez.

AT : J'existe pour faciliter l'écriture des jointures.

AC : Et vous m'aidez dans ces mêmes jointures, AT.

TLP : Je ne comprends pas. Pouvez-vous me montrer un exemple ?

AT : Je peux déjà vous montrer la syntaxe. Je pense que ce que je fais y est assez clair :

```
SELECT mc.nom, mc.prenom, p.profession

FROM mes_contacts AS mc

  INNER JOIN

  profession AS p

WHERE mc.id_prof = p.id_prof;
```

TLP : Je vous vois ! Partout où je devrais taper `mes_contacts`, je peux simplemment taper `mc` à la place. Et `p` pour `profession`. Beaucoup plus simple. Et vraiment utile quand je dois inclure deux noms de tables dans une seule requête.

AT : Spécialement quand les tables ont des noms qui se ressemblent. Faciliter la compréhension de vos requêtes non seulement aide à les écrire, mais cela vous aide aussi à vous souvenir de ce qu'elles font quand vous y revenez plus tard.

TLP : Merci beaucoup, AT et AC. Ce fut.. euh... où sont-ils partis ?

Révélations sur les alias de tables et de colonnes

L'interview de la semaine : De quoi vous cachez-vous ?

Tête La Première : Bienvenue à Alias de Table et Alias de Colonne. Nous sommes heureux que vous puissiez être là tous les deux. Nous espérons que vous pouvez éclaircir certains points confus pour nous.

Alias de Table : Certainement, content d'être là. Et vous pouvez nous appeler **AT** et **AC** pendant cette interview, c'est plus court [rires].

TLP : Ha ha! En tout cas, c'est de circonstance. D'accord, **AC**, commençons par vous. Pourquoi tous ces mystères ? Essayez-vous de cacher quelque chose ?

Alias de Colonne : Absolument pas ! Au mieux, j'essaie de rendre les choses plus claires. Je pense que je parle en nos deux noms là, n'est-ce pas AT ?

AT : Tout à fait. Dans le cas d'AC, ce qu'il fait doit déjà être évident. Il prend des noms de colonnes longs et redondants et les rend plus faciles à suivre. Plus accessibles. Il vous donne aussi les tables de résultats avec des noms de colonnes utiles. Pour moi, c'est un peu différent.

TLP : Je dois admettre que je ne vous connais pas très bien, AT. Je vous ai vu à l'oeuvre, mais je ne suis toujours pas certain de ce que vous faites. Vous n'apparaissez pas du tout dans les résultats quand nous vous utilisons dans une requête.

AT : Oui, c'est vrai. Mais je pense que vous ne saisissez pas encore ma vocation supérieure.

TLP : Vocation supérieure ? Ça a l'air fascinant. Continuez.

AT : J'existe pour faciliter l'écriture des jointures.

AC : Et vous m'aidez dans ces mêmes jointures, AT.

TLP : Je ne comprends pas. Pouvez-vous me montrer un exemple ?

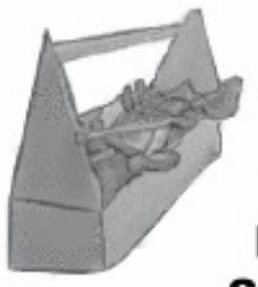
AT : Je peux déjà vous montrer la syntaxe. Je pense que ce que je fais y est assez clair :

```
SELECT mc.nom, mc.prenom, p.profession

FROM mes_contacts AS mc

  INNER JOIN

  profession AS p

WHERE mc.id_prof = p.id_prof;
```

TLP : Je vous vois ! Partout où je devrais taper `mes_contacts`, je peux simplemment taper `mc` à la place. Et `p` pour `profession`. Beaucoup plus simple. Et vraiment utile quand je dois inclure deux noms de tables dans une seule requête.

AT : Spécialement quand les tables ont des noms qui se ressemblent. Faciliter la compréhension de vos requêtes non seulement aide à les écrire, mais cela vous aide aussi à vous souvenir de ce qu'elles font quand vous y revenez plus tard.

TLP : Merci beaucoup, AT et AC. Ce fut.. euh... où sont-ils partis ?



Boîte à outils SQL

Vous venez de terminer le chapitre 8 et pouvez utiliser JOIN comme un vrai pro du SQL. Regardez toutes les techniques que vous avez apprises. Pour une liste exhaustive des outils du livre, voyez l'annexe C.

INNER JOIN : $

Une jointure qui combine les enregistrements de deux tables en utilisant une condition.

NATURAL JOIN : JOINTURE NATURELLE

Une jointure interne sans la clause « ON ». Cela ne fonctionne que si vous joignez deux tables qui ont un même nom de colonne.

ÉQUI-JOINTURE et NON ÉQUI-JOINTURE

Des jointures internes toutes les deux. L'ÉQUI-JOINTURE renvoie des lignes qui sont égales et la NON ÉQUI-JOINTURE renvoie toutes les lignes qui ne sont pas égales.

CROSS JOIN : JOINTURE CROISÉE

Renvoie toutes les lignes d'une table, croisées avec toutes les lignes d'une seconde table. Connue sous beaucoup d'autres noms y compris JOINTURE CARTÉSIENNE et NON-JOINTURE.

JOINTURE VIRGULE

La même chose qu'une JOINTURE CROISÉE, sauf qu'on utilise une virgule au lieu des mots-clés CROSS JOIN.

À vos crayons ! Solution

De la page 348.

Vous savez déjà comment ALTERer des tables. Vous devez ALTERer mes_contacts pour obtenir quatre nouvelles colonnes. Appelez-les centre_interet1, centre_interet2, centre_interet3 et centre_interet4.

```
ALTER TABLE mes_contacts
ADD (centre_interet1 VARCHAR(30), centre_interet2 VARCHAR(30), centre_
interet3 VARCHAR(30), centre_interet4 VARCHAR(30));
```

À vos crayons ! Solution

De la page 350.

Complétez les blancs pour terminer l'instruction update de Greg. Nous avons ajouté quelques remarques pour vous aider.

La différence entre SUBSTRING_INDEX et SUBSTR est que SUBSTRING_INDEX cherche une chaîne *à l'intérieur* de la colonne centres_interet (dans ce cas, une virgule) et renvoie tout ce qui se trouve devant. SUBSTR raccourcit la longueur de la colonne centres_interet (en faisant commencer la chaîne juste après le premier centre d'intérêt, une virgule et un espace (d'où le +2)).

```
UPDATE mes_contacts SET
centre_interet1 = SUBSTRING_INDEX(centres_interet, ',', 1),
centres_interet = SUBSTR(centres_interet, LENGTH(centre_interet1)+2),
centre_interet2 = SUBSTRING_INDEX( centres_interet, ',', 1 ),
centres_interet = SUBSTR( centres_interet, LENGTH(centre_interet2)+2 ),
centre_interet3 = SUBSTRING_INDEX( centres_interet, ',', 1 ),
centres_interet = SUBSTR( centres_interet, LENGTH(centre_interet3)+2 ),
centre_interet4 = centres_interet;
```

Après avoir enlevé les trois premiers centres d'intérêt de la colonne centres_interet, il ne reste plus que le quatrième centre d'intérêt. Cette ligne le déplace simplement dans la nouvelle colonne. Nous aurions aussi pu simplement renommer la colonne centres_interet en centre_interet4 à la place.

Il reste le quatrième centre d'intérêt dans la colonne centres_interet après que nous ayons exécuté la commande.

centres_interet	centre_interet1	centre_interet2	centre_interet3	centre_interet4
deuxième, troisième, quatrième	premier	deuxième	troisième	quatrième

9 sous-requêtes

Des requêtes dans les requêtes

Oui, Jacky, je voudrais une question en deux parties s'il te plaît. Les jointures sont géniales, mais parfois vous avez ***plus d'une question à poser à votre base de données***. Ou vous devez ***prendre le résultat d'une requête pour l'utiliser dans la formulation d'une autre requête***. C'est là que les **sous-requêtes** entrent en jeu. Elles vous aident à **éviter les données dupliquées, elles dynamisent vos requêtes** et elles vous font même entrer dans toutes ces somptueuses fêtes d'après-concert. (En fait, pas vraiment, mais deux sur trois, c'est pas mal !)

Greg débarque sur le marché du recrutement

Jusqu'ici, la base de données `liste_greg` a été, littéralement, le fruit d'une passion. Elle a aidé Greg à trouver des rendez-vous pour ses amis, mais il n'en a pas retiré d'argent.

L'idée lui est venue de lancer une entreprise de recrutement où il ferait correspondre ses contacts avec des emplois possibles.

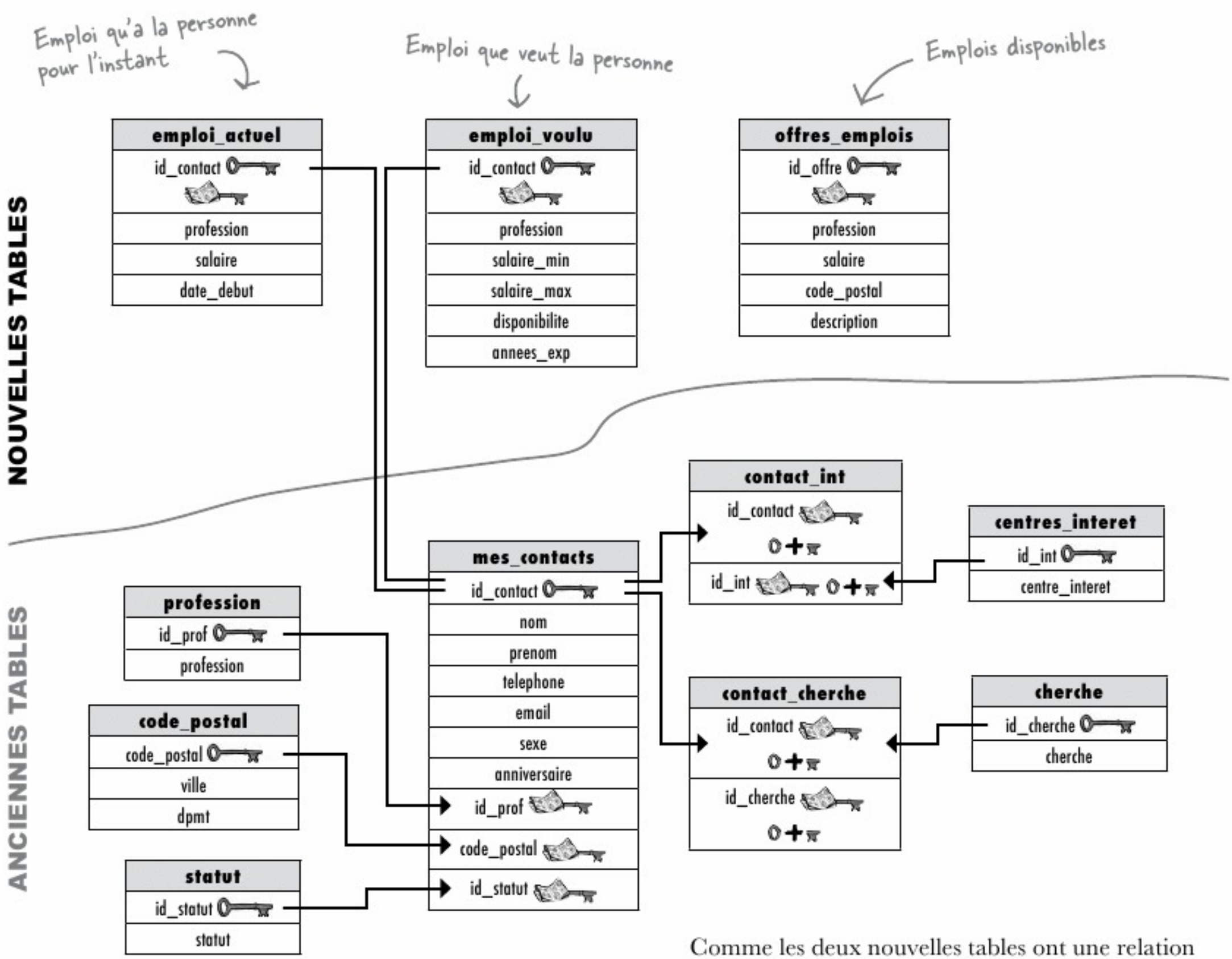
Greg sait qu'il va devoir ajouter de nouvelles tables pour ses contacts intéressés par le service. Il décide de créer des tables un-à-un séparées, plutôt que de mettre ces informations dans `mes_contacts`, pour deux raisons.

D'abord, dans sa liste `mes_contacts`, tout le monde n'est pas intéressé par ce service. Ainsi, il évite des valeurs `NULL` dans `mes_contacts`.

Ensuite, il est possible qu'un jour il embauche des gens pour l'aider dans son entreprise et les informations sur les salaires peuvent être considérées comme sensibles. **Il ne voudra peut-être donner accès à ces tables qu'à certaines personnes**.

La liste de Greg reçoit des tables supplémentaires

Greg a ajouté de nouvelles tables à sa base de données pour conserver les informations sur la **position voulue** et **la tranche de salaire espérée**, ainsi que la **position** et le **salaire actuels**. Il a aussi créé une simple table pour conserver les **informations sur les offres d'emplois**.



Comme les deux nouvelles tables ont une relation **un-à-un** avec `mes_contacts`, il a pu, jusqu'à maintenant, utiliser des **jointures naturelles** avec grande facilité et grand succès.

Greg utilise une jointure interne

Greg a des annonces d'emplois très intéressantes et il essaie de trouver les personnes correspondantes dans sa base de données. Il veut trouver la meilleure personne pour cet emploi parce qu'il reçoit une prime si son candidat est engagé.

Cherchons : Développeur web

Recherchons développeur web avec code HTML et CSS de grande qualité pour travailler avec nos équipes d'interaction et de conception visuelle. Excellente opportunité de briller avec une entreprise très connue pour une personne méticuleuse sur les standards web. Travail avec une entreprise extrêmement influente dirigée par des personnes intelligentes qui adorent ce qu'elles font.

Salaire : 95 000€ - 105 000€

Experience : 5 années minimum

Une fois qu'il aura trouvé les meilleures correspondances, il pourra les appeler pour filtrer davantage. Mais d'abord, il veut trouver tous lesdéveloppeurs web avec au moins cinq ans d'expérience n'exigeant pas de salaire supérieur à 105,000.

À vos crayons !

Écrivez une requête pour trouver les candidats qualifiés de la base de données.

emploi_actuel
id_contact
profession
salaire
date_debut

emploi_voulu
id_contact
profession
salaire_min
salaire_max
disponibilite
annees_exp

offres_emplois
id_offre
profession
salaire
code_postal
description

Voici le salaire le plus bas qu'ils peuvent accepter pour un nouvel emploi.

Voici le salaire qu'ils espèrent pour un nouvel emploi.

Mais il veut essayer d'autres requêtes

Greg a plus d'offres d'emploi qu'il n'en faut. Il va chercher des gens dans sa table profession pour voir s'il peut trouver des correspondances pour ses offres d'emploi. Ensuite il pourra faire une jointure naturelle avec `mes_contacts` pour les contacter et voir s'ils sont intéressés.

Il sélectionne d'abord toutes les professions de sa table offres_emplois.

```
SELECT profession FROM offres_emplois
GROUP BY profession ORDER BY profession;
```

Nous utilisons GROUP BY pour n'obtenir qu'une ligne par profession. Nous les classons aussi par ordre alphabétique.

Les résultats

profession
Coiffeur
Concepteur web
Cuisinier
Développeur web
Serveur

Voici quelques-unes des professions de la table offres_emplois de Greg.

À vos crayons !

Écrivez une requête pour trouver les candidats qualifiés de la base de données.

```
SELECT mc.nom, mc.prenom, mc.telephone
FROM mes_contacts AS mc
    NATURAL JOIN
    emploi_voulu AS ev
WHERE ev.profession = 'Développeur web'
AND ev.salaire_min < 105000;
```

Nous n'avons besoin que des informations pour les contacter puisque nous savons déjà qu'ils cherchent un emploi de développeur web.

Puisque mes_contacts et emploi_voulu partagent tous les deux id_contact comme clé primaire, nous pouvons utiliser une simple jointure naturelle pour les relier.

Nous nous intéressons seulement aux gens qui vont prendre ce salaire en considération. Nous regardons le salaire_min pour voir si le salaire offert est supérieur au minimum qu'ils sont prêts à accepter.

Et maintenant Greg utilise le mot-clé IN pour voir s'il a des correspondances pour ces professions parmi ses contacts.

```
SELECT mc.prenom, mc.nom, mc.telephone, ea.profession
FROM emploi_actuel AS ea NATURAL JOIN mes_contacts AS mc
WHERE
ea.profession IN ('Coiffeur', 'Concepteur web', 'Cuisinier', 'Développeur web', 'Serveur');
```

Vous souvenez-vous du mot-clé IN ? Il renvoie une ligne si ea.profession est dans le groupe de professions entre parenthèses.

Résultats de la première requête

mc.prenom	mc.nom	mc.telephone	ea.profession
Mathieu	Delecroix	0655532140	Cuisinier
Céline	Dupré	0655589760	Serveur
Cyril	Dumoulin	0655544430	Concepteur web
Frédéric	Lallemand	0655556740	Développeur web
Germain	Bonte	0655500980	Développeur web

Ça marche !

Mais il doit toujours taper deux requêtes séparées...

Essayez de combiner les deux requêtes en une seule et écrivez ici cette requête unique.

Sous-requêtes

Pour accomplir ce que font ces deux requêtes en une seule requête, nous devons ajouter une **sous-requête** dans la requête.

Nous allons appeler la *deuxième* requête, celle que nous avons utilisée pour obtenir les correspondances de la table `professions`, la requête `EXTERNE` parce qu'elle va envelopper la requête `INTERNE`. Voyons comment cela fonctionne :

requête EXTERNE

```
SELECT mc.prenom, mc.nom, mc.telephone, ea.profession
FROM emploi_actuel AS ea NATURAL JOIN mes_contacts AS mc
WHERE
ea.profession IN ('Coiffeur', 'Concepteur web', 'Cuisinier', 'Développeur web', 'Serveur');
```

Cette partie est la requête externe.

Cette partie peut être enlevée et remplacée avec une partie de notre première requête, qui va devenir la requête interne.

Toutes ces professions entre parenthèses ci-dessus viennent de la *première* requête que nous avons faite, celle qui sélectionne toutes les professions de la table `offres_emplois`. Donc (et voici la partie intéressante, faites bien attention) nous pouvons **remplacer cette partie de la requête externe** avec **une partie de notre première requête**. Celle-ci produira toujours tous les résultats entre parenthèses ci-dessus, mais cette requête est maintenant encapsulée comme sous-requête :

Une sous-requête est une requête qui est enveloppée par une autre requête. On l'appelle aussi une requête INTERNE.

requête INTERNE

```
SELECT profession FROM offres_emplois
GROUP BY profession ORDER BY profession;
```

Cette partie de la première requête va devenir la requête interne ou sous-requête.

Nous combinons les deux en une requête avec sous-requête

Nous avons simplement combiné les deux requêtes en une. La première requête s'appelle la **requête externe.** Celle qui se trouve à l'intérieur s'appelle la **requête interne**.

requête EXTERNE + requête INTERNE

= Requête avec une sous-requête

Les deux requêtes, combinées en une, forment une requête contenant une sous-requête.

Requête externe

```
SELECT mc.prenom, mc.nom, mc.telephone, ea.profession
FROM emploi_actuel AS ea NATURAL JOIN mes_contacts AS mc
WHERE ea.profession IN (SELECT profession FROM offres_emplois);
```

Nous ne devons plus retaper toutes les professions de notre première requête parce que la requête interne s'en occupe pour nous !

Et voici les résultats que nous obtenons quand nous exécutons notre requête : exactement les mêmes que quand nous avons réécrit ***toutes*** les professions dans la clause `WHERE`, mais avec beaucoup moins de frappe.

Mêmes résultats qu'auparavant, mais avec une seule requête !

mc.prenom	mc.nom	mc.telephone	ea.profession
Mathieu	Delecroix	0655532140	Cuisinier
Céline	Dupré	0655589760	Serveur
Cyril	Dumoulin	0655544430	Concepteur web
Frédéric	Lallemand	0655556740	Développeur web
Germain	Bonte	0655500980	Développeur web

Anatomie d'une requête à l'intérieur d'une requête

Comme si une requête ne suffisait pas : voici la sous-requête

Une sous-requête n'est rien de plus qu'une requête à l'intérieur d'une autre requête.

La requête à l'extérieur s'appelle la **requête contenante,** ou **requête externe**. La requête à l'intérieur est la **requête interne**, ou **sous-requête**.

Requête externe. Parfois appelée requête contenante.

```
SELECT une_colonne, une_autre_colonne
FROM table
WHERE colonne = (SELECT colonne FROM table);
```

Requête interne ou sous-requête.

Requête externe

```
SELECT une_colonne, une_autre_colonne
FROM table
WHERE colonne = (SELECT colonne FROM table);
```

Requête interne

Parce qu'elle utilise l'opérateur =, cette sous-requête va renvoyer une ***valeur unique***, **une ligne d'une colonne** (parfois appelée une *cellule*, mais dénominée *valeur scalaire* en SQL), qui est comparée aux colonnes de la clause WHERE.

valeur

Notre sous-requête renvoie une valeur scalaire (une colonne, une ligne), qui est ensuite comparée aux colonnes de la clause WHERE.

Une sous-requête au travail

Voyons une requête semblable à l'œuvre dans la table `mes_contacts`. D'abord votre SGBDR prend la valeur scalaire de la table `code_postal`, puis il compare cette valeur ç colonnes de la clause `WHERE`.

```
(SELECT code_postal FROM
code_postal WHERE ville =
'Dijon' AND dpmt = '21')
```

valeur

```
SELECT nom, prenom
FROM mes_contacts
WHERE code_postal = (SELECT code_postal FROM
                     code_postal WHERE ville =
                     'Dijon' AND dpmt = '21')
```

Cette requête sélectionne les noms de gens de mes_contacts à Dijon, Côte d'or.

il n'y a pas de

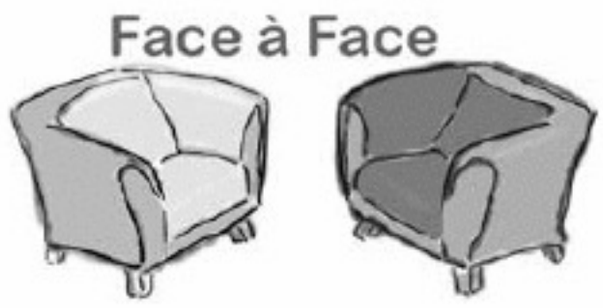
Questions stupides

Q: Pourquoi ne puis-je pas faire ça avec une jointure ?

R: Vous pouvez, mais certains trouvent les sous-requêtes plus faciles à écrire que les jointures. C'est mieux d'avoir le choix de la syntaxe.

Vous pouvez faire la même requête que ci-dessus de cette façon :

```
SELECT nom, prenom
FROM mes_contacts mc
NATURAL JOIN code_postal cp
WHERE cp.ville = 'Dijon'
AND cp.dpmt = '21'
```



Le débat de la soirée : **Êtes-vous INTERNE ou EXTERNE ?**

Requête externe

Vous savez, Requête interne, je n'ai pas vraiment besoin de vous. J'irais très bien sans vous.

Hourra pour la Requête interne ! Vous me donnez un tout petit résultat. Les utilisateurs veulent des données, beaucoup de données. Je les leur donne. Mmm, je parie que si vous n'étiez pas là, ils seraient encore plus contents.

Pas si j'ajoutais une clause WHERE.

Oh que si. À quoi peut servir une réponse d'une ligne, d'une colonne ? Il n'y a pas assez d'information.

Bien sûr, mais je suis indépendante.

Requête interne

Je pourrais me débrouiller toute seule également. Croyez-vous que c'est amusant de vous donner un résultat ciblé et spécifique, tout cela pour que vous en tiriez un tas de lignes correspondantes ? La quantité n'est pas la qualité, vous savez.

Non, je donne à vos résultats une sorte de raison d'être. Sans moi, vous recracheriez toutes les données de la table.

Justement, JE SUIS votre clause WHERE. Et une clause très spécifique, si je puis me permettre. En fait, je n'ai pas vraiment besoin de vous.

Alors peut-être que nous travaillons bien ensemble. Je donne une direction à vos résultats.

Tout comme moi, la plupart du temps.

Les règles des sous-requêtes

Toutes les sous-requêtes suivent certaines règles. Complétez les blancs en utilisant les mots ci-dessous (vous pouvez en utiliser certains plusieurs fois).

POINT-VIRGULE
SELECT
LISTE DES COLONNES
FIN
PARENTHÈSES
DELETE
UPDATE
INSERT
HAVING
FROM

Les règles SQL de l'ordre

Une sous-requête est toujours une instruction . ..... unique.

Les sous-requêtes sont toujours entre . ...

Les sous-requêtes n'ont pas leurs propres . ...
Comme toujours, un . ... va à la . .....de la requête tout entière.

Les règles SQL de l'ordre

Les sous-requêtes peuvent apparaître à quatre endroits dans une requête : une clause . ..... , la . ... du SELECT en tant qu'une des colonnes, une clause . ..... et dans une clause . ...

Les sous-requêtes peuvent être utilisées avec . ..... , ... , ... et, bien sûr, ...

Les règles des sous-requêtes

Gardez ces règles à l'esprit en regardant les sous-requêtes dans le reste du chapitre.

Les règles SQL de l'ordre

Une sous-requête est toujours une instruction **SELECT** unique.

Les sous-requêtes sont toujours entre **PARENTHÈSES**.

Les sous-requêtes n'ont pas leurs propres **POINT-VIRGULES**. Comme toujours, un **POINT-VIRGULE** va à la **FIN** de la requête tout entière.

Les règles SQL de l'ordre

Les sous-requêtes peuvent apparaître à quatre endroits dans une requête : une clause **SELECT**, la **LISTE DES COLONNES** du SELECT en tant qu'une des colonnes, une clause **FROM** et dans une clause **HAVING**.

Les sous-requêtes peuvent être utilisées avec **INSERT**, **DELETE**, **UPDATE** et, bien sûr, **SELECT**.

il n'y a pas de

Questions stupides

Q: Alors qu'est-ce que la requête interne a le droit de renvoyer ? Et la requête externe ?

R: Dans la plupart des cas, la requête interne ne peut renvoyer qu'une valeur unique, c'est-à-dire une colonne d'une ligne. La requête externe peut ensuite utiliser cette valeur pour la comparer avec toutes les valeurs d'une colonne.

Q: Pourquoi dites-vous « une valeur unique » alors que l'exemple de la page 388 renvoie une colonne pleine de valeurs ?

R: Parce que l'opérateur `IN` regarde un ensemble de valeurs. Si vous utilisez un opérateur de comparaison, comme = dans l'Anatomie, vous ne pouvez avoir qu'une valeur à comparer à chaque valeur de votre colonne.

Q: Je ne comprends toujours pas si une sous-requête peut renvoyer une valeur unique ou plus d'une valeur. Quelles sont les règles officielles ?

R: En général, une sous-requête doit renvoyer une valeur unique. `IN` est l'exception. Le plupart du temps les sous-requêtes doivent renvoyer une valeur unique pour fonctionner.

Q: Alors que se passe-t-il si votre sous-requête renvoie *vraiment* plus d'une valeur mais que la requête externe n'utilise pas de clause WHERE contenant un ensemble de valeurs ?

R: Le chaos ! Une destruction de masse ! En fait, vous aurez seulement une erreur.

Oui, ces règles sont sympas ou ce que vous voulez, mais ce que je veux savoir, c'est comment me débarrasser de ces longs noms dans mes colonnes de résultats, comme mc.code_postal. Avez-vous une règle pour cela ?

En fait, il y a deux choses que vous pouvez faire pour vous aider à réduire l'encombrement.

Vous pouvez créer des alias pour vos colonnes dans la liste de vos colonnes SELECT. La table que vous obtenez avec vos résultats est soudain beaucoup plus claire.

Voici la sous-requête que nous venons de créer, mais avec de courts **alias de colonnes**.

Dans nos résultats, nous allons donner l'alias 'prenom' à la colonne prenom de mes_contacts.

... et la colonne nom de mes_contacts aura l'alias 'nom' dans nos résultats.

```
SELECT mc.prenom AS prenom, mc.nom AS nom, mc.telephone AS
tel, ea.profession AS emploi

FROM emploi_actuel AS ea NATURAL JOIN mes_contacts AS mc

WHERE emploi IN (SELECT profession FROM offres_emplois);
```

La colonne telephone de mes_contacts va avoir l'alias 'tel' dans nos résultats... etc. Vous avez compris !

Souvenez-vous, le mot-clé AS est optionnel, vous pouvez donc l'omettre quand vous créez vos alias.

Voici les résultats que vous donne la requête.

Remarquez qu'utiliser des alias de colonnes facilite la compréhension des résultats.

Et comme les alias sont temporaires, nous n'affectons aucun des noms de colonnes ou de tables dans aucune des tables.

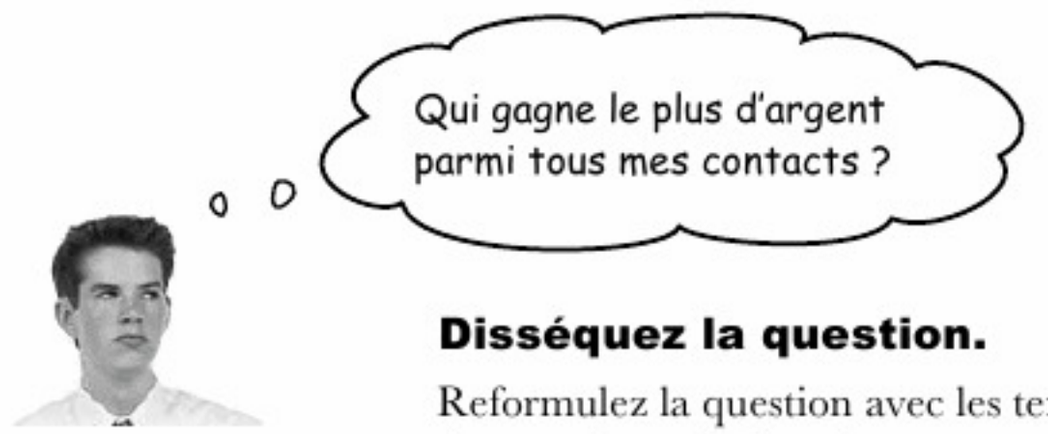
prenom	nom	tel	emploi
Mathieu	Delecroix	0655532140	Cuisinier
Céline	Dupré	0655589760	Serveur
Cyril	Dumoulin	0655544430	Concepteur web
Frédéric	Lallemand	0655556740	Développeur web
Germain	Bonte	0655500980	Développeur web

Visite d'une sous-requête en construction

Le plus difficile avec les sous-requêtes n'est pas la structure ; c'est de comprendre quelle partie de la requête doit être la sous-requête. Voire même si vous avez vraiment besoin d'une sous-requête.

Analyser des requêtes est similaire à résoudre une charade. Vous identifiez les mots de la question qui correspondent aux choses que vous connaissez (comme les noms de colonnes et de tables) et vous séparez les choses.

Parcourons l'analyse d'une question que nous voulons poser à notre base de données et voyons comment en faire une requête. D'abord, la question :



Disséquez la question.

Reformulez la question avec les termes des colonnes et des tables de votre base de données.

« Qui » signifie que vous voulez un prénom et un nom de `mes_contacts`. « Le plus d'argent » signifie que vous avez besoin d'une valeur `MAX` de votre table `emploi_actuel`.

Qui gagne le plus d'argent parmi tous mes contacts ?

Prenom et nom de mes_contacts

MAX(salaire) de la table emploi_actuel

Identifiez une requête qui répond à une partie de la question.

Puisque nous créons une sous-requête non-corrélée, nous pouvons diviser notre question et construire une requête qui y réponde en partie.

Ce `MAX(salaire)` semble un bon candidat pour notre première requête.

```
SELECT MAX(salaire) FROM emploi_actuel;
```

Vous souvenez-vous de MAX ? Il renvoie la plus grande valeur de la colonne entre parenthèses.

Continuez à disséquer votre requête.

La première partie de la requête est également facile ; nous devons simplement sélectionner le nom et prénom :

```
SELECT mc.prenom, mc.nom
FROM mes_contacts AS mc;
```

SELECTionne le prénom et le nom.

Enfin, trouvez comment les relier.

Nous n'avons pas seulement besoin des noms des personnes de `mes_contacts`, nous devons aussi connaître leurs salaires pour pouvoir les comparer à notre `MAX(salaire)`. Il nous faut une jointure interne naturelle pour renvoyer le salaire de chacun :

```
SELECT mc.prenom, mc.nom,
ea.salaire
FROM mes_contacts AS mc
NATURAL JOIN emploi_actuel AS
ea;
```

Utilise NATURAL JOIN pour récupérer le salaire de chacun.

Et maintenant ajoutez la clause `WHERE` pour les relier.

Nous créons une grande requête qui répond à la question : « Qui gagne le plus d'argent ? »

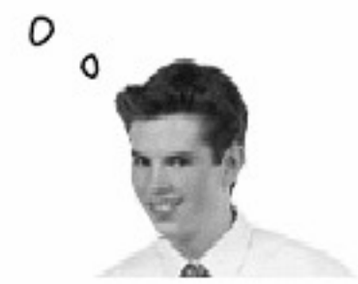
Voici la partie que nous venons d'écrire : elle renvoie le salaire de chaque personne.

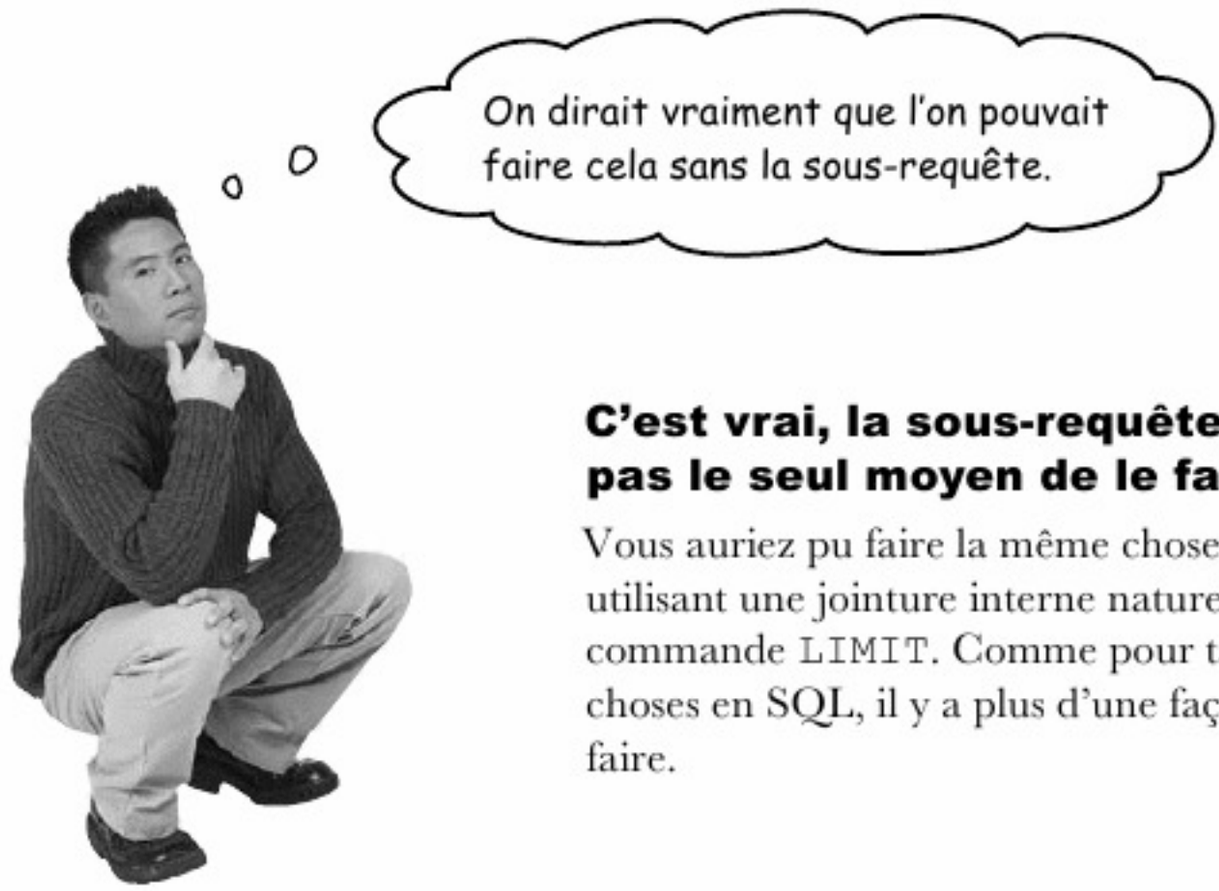
```
SELECT mc.prenom, mc.nom, ea.salaire
FROM mes_contacts AS mc NATURAL JOIN emploi_actuel AS ea
WHERE ea.salaire =
(SELECT MAX(ea.salaire) FROM emploi_actuel ea);
```

Et voici la première partie qui est maintenant notre sous-requête pour trouver la valeur MAX de salaire. Cette valeur est comparée à la partie externe de la requête pour donner le résultat.

mc.prenom	mc.nom	ea.salaire
Benoît	Lemaire	187000

C'est Benoît ? J'aurais dû m'en douter. Il n'offre jamais de payer l'addition.



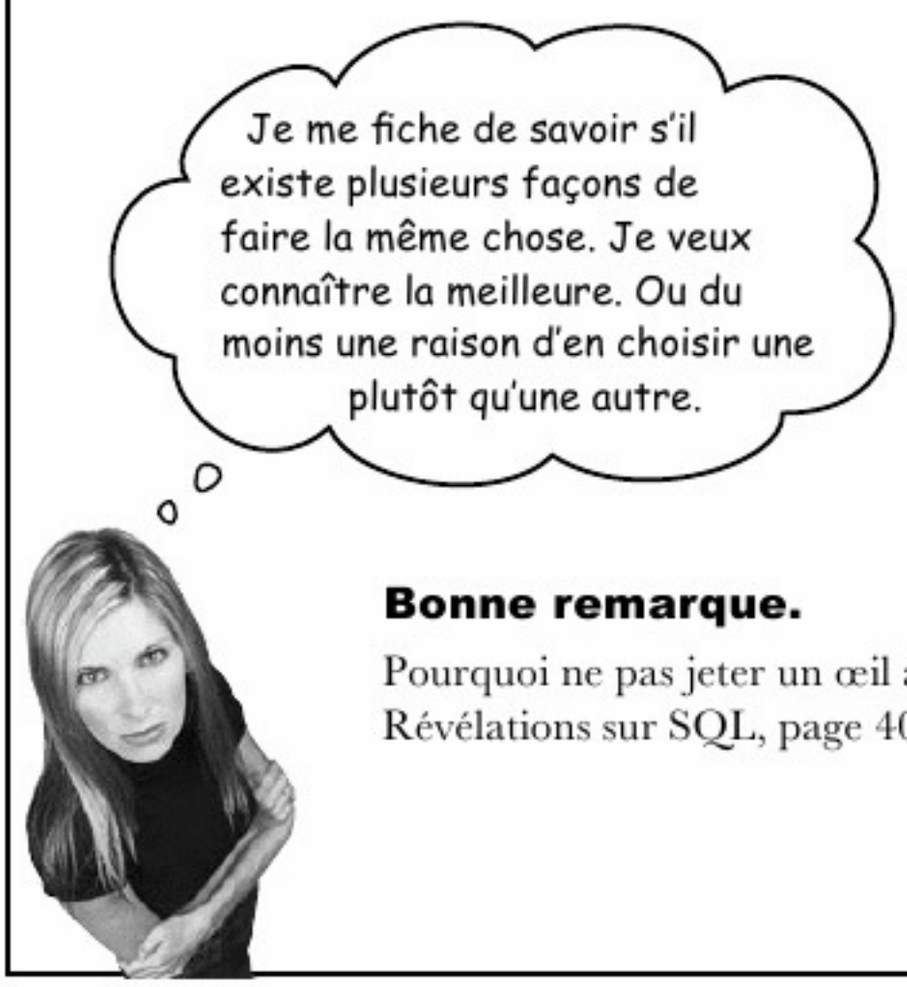


C'est vrai, la sous-requête n'était pas le seul moyen de le faire.

Vous auriez pu faire la même chose en utilisant une jointure interne naturelle et une commande `LIMIT`. Comme pour tant d'autres choses en SQL, il y a plus d'une façon de le faire.

MUSCLEZ VOS NEURONES

Écrivez une autre requête pour trouver qui gagne le plus d'argent de tous les contacts de Greg.



Bonne remarque.

Pourquoi ne pas jeter un œil à l'interview Révélations sur SQL, page 400 ?

Une sous-requête comme colonne de SELECT

Une sous-requête peut être utilisée comme l'une des colonnes d'une instruction SELECT. Prenez cette requête.

```
SELECT mc.prenom, mc.nom,
(SELECT dpmt
FROM code_postal
WHERE mc.code_postal = code_postal) AS dpmt
FROM mes_contacts mc;
```

Nous définissons un alias de colonne, 'dpmt'.

Nous pouvons disséquer cette requête en commençant par regarder la sous-requête. La sous-requête fait simplement correspondre les codes postaux aux départements correspondants dans la table `code_postal`.

En termes plus simples, voici ce que fait cette requête :

> Passe dans toutes les lignes de la table mes_contacts. Pour chacune d'entre elles, récupère le prénom, nom et département (le département étant trouvé en prenant le code postal et en le faisant correspondre au département dans la table code_postal).

Rappelez-vous qu'une sous-requête ne peut renvoyer qu'une seule valeur, donc à chaque fois qu'elle s'exécute, une ligne est renvoyée. Voici à quoi peuvent ressembler quelques-uns des résultats de cette requête :

mc.prenom	mc.nom	dpmt
Mathieu	Delecroix	31
Céline	Dupré	72
Cyril	Dumoulin	22
Frédéric	Lallemand	65
Germain	Bonte	58

Si une sous-requête est utilisée comme expression d'une colonne dans une instruction SELECT, elle ne peut renvoyer qu'une valeur d'une colonne.

Un autre exemple : sous-requête avec une jointure naturelle

Arnaud, l'ami de Greg, s'est vanté de recevoir un très gros salaire. Il n'a pas dit combien à Greg, mais ce dernier pense avoir cette information dans sa table. Il fait un rapide `NATURAL JOIN` pour la trouver, en utilisant l'adresse email d'Arnaud.

```
SELECT ea.salaire
FROM mes_contacts mc NATURAL JOIN emploi_actuel ea
WHERE email = 'arnaud@stationmeteo.com';
```

Cette requête va renvoyer le salaire d'Arnaud, une valeur unique.

Ceci sera la requête interne.

Greg remarque que cette requête ne va renvoyer qu'un seul résultat. Plutôt que de l'exécuter, d'obtenir cette valeur et de la balancer dans une autre requête, il décide d'en faire une sous-requête.

Il écrit une requête unique pour :

- récupérer le salaire d'Arnaud,
- le comparer à d'autres salaires
- et renvoyer les noms, prénoms et salaires des gens
- qui gagnent plus qu'Arnaud.

Ici on utilisera l'opérateur de comparaison >.

Les salaires supérieurs à celui d'Arnaud.

C'est une longue requête, mais elle me permet de comparer quelque chose à d'autres choses de ma base de données **sans avoir besoin de la connaître**.

Voici la requête externe

```
SELECT mc.prenom, mc.nom, ea.salaire
FROM
mes_contacts AS mc NATURAL JOIN emploi_actuel AS ea
WHERE
ea.salaire > (LA REQUÊTE DU SALAIRE D'ARNAUD);
```

Une sous-requête non-corrélée

Quand nous assemblons le tout, voici la requête en entier. Le logiciel traite d'abord la requête interne, puis il utilise cette valeur pour trouver le résultat de la requête externe.

Récupère le prénom, le nom et le salaire.

```
SELECT mc.prenom, mc.nom, ea.salaire
FROM
mes_contacts AS mc NATURAL JOIN emploi_actuel AS ea
WHERE
ea.salaire > (SELECT ea.salaire
FROM mes_contacts mc NATURAL JOIN emploi_actuel ea
WHERE email = 'arnaud@stationmeteo.com');
```

Le SGBDR exécute cette partie en deuxième.

Ne montre que les gens qui ont un plus grand salaire que celui d'Arnaud.

Ces deux requêtes sont exécutées séparément par le SGBDR.

La sous-requête qui récupère le salaire d'Arnaud pour que la requête externe puisse la comparer.

Ceci est traité en premier.

Voici quelques résultats. Nous n'avons pas utilisé ORDER BY, ils ne sont donc pas classés.

mc.prenom	mc.nom	ea.salaire
Angelo	Cartier	46500
Martin	Duchemin	78000
Caroline	Semelle	48000
Lucien	Dugenou	49000
Julie	Mercier	120000

Toutes les sous-requêtes que vous avez vues jusqu'ici s'appellent des **sous-requêtes non-corrélées**. La requête interne est exécutée en premier, puis le résultat est utilisé dans la condition WHERE de la requête externe. ***Mais la requête interne ne dépend en aucune façon des valeurs de la requête externe ; elle peut être exécutée comme requête indépendante.***

Si la sous-requête est indépendante et ne fait pas référence à la requête externe, c'est une sous-requête non-corrélée.

(et si vous réussissez à placer « sous-requête non-corrélée » dans une conversation, les non-utilisateurs d'SQL seront très impressionnés)

Le requête externe est traitée en second. Ses résultats dépendent de la valeur de la requête interne.

La requête interne est indépendante et est traitée en premier.

Révélations sur SQL

L'interview de la semaine : Choisir la meilleure requête

SQL Tête la première : Bienvenue, SQL. Nous apprécions l'interview personnelle. Nous savons que les temps sont difficiles.

SQL : Difficiles ? C'est ce que vous en dites ? Je dirais que les temps sont perturbants, dérangeants et très difficiles à quantifier tout en étant, en même temps, très tortueux.

SQL TLP : Euh, tout à fait. C'est plus ou moins notre sujet ici. Vous recevez des plaintes disant que vous êtes peut-être trop souple. Vous nous donnez trop de choix quand on vous pose des questions.

SQL : J'admets être souple. Que vous pouvez me poser la même question de différentes façons et que je vous donnerai les mêmes réponses.

SQL TLP : Certains diraient que vous ne savez pas ce que vous voulez.

SQL : Je refuse de me défendre de cela. Je ne suis pas le méchant ici.

SQL TLP : Non, nous le savons, c'est juste que vous êtes si…imprécis.

SQL : HA ! Moi imprécis ! J'ai eu ma dose de tout ceci. (se lève)

SQL TLP : Non, ne partez pas. Nous voulons seulement quelques réponses. Parfois vous nous laissez vous demander la même chose de tellement de façons différentes.

SQL : Et qu'y a-t-il de mal à cela ?

SQL TLP : En fait rien, nous voulons seulement savoir QUOI vous demander. Cela a-t-il une importance, si vous nous donnez les mêmes réponses ?

SQL : Bien sûr que c'est important ! Vous me demandez parfois quelque chose et cela me prend très longtemps de répondre. Parfois, POUF, j'ai fini. L'important est que vous me le demandiez de la bonne façon.

SQL TLP : Alors c'est à propos du temps qu'il vous faut pour répondre ? C'est comme cela que l'on choisit la façon de vous interroger ?

SQL : Hé bien oui. Bien sûr que oui. Tout dépend de ce que vous me demandez. Je suis seulement là pour essayer de répondre à vos questions, si elles sont précises.

SQL TLP : La vitesse ? C'est ça le secret ?

SQL : Écoutez, je vais vous donner un coup de main. Le problème avec les bases de données est qu'elles GRANDISSENT. Vous voulez que vos questions soient aussi faciles à répondre que possible. Parce que si vous me demandez « Qui a tué le colonnel Moutarde » vous devez m'y faire réfléchir aussi peu que possible. Donnez-moi des questions faciles et je vous donnerai des réponses rapides.

SQL TLP : Je comprends. Mais comment puis-je savoir quelles sont les questions faciles ?

SQL : Bon, pour commencer, les jointures croisées sont un énorme gaspillage de temps. Et les sous-requêtes corrélées sont aussi plutôt lentes.

SQL TLP : Autre chose ?

SQL : Hé bien…

SQL TLP : S'il vous plaît, continuez.

SQL : Expérimenter. Parfois votre meilleure option est de créer des tables de test et d'essayer différentes requêtes. Puis vous pouvez comparer le temps qu'a pris chacune. Oh, et les jointures sont plus efficaces que les sous-requêtes.

SQL TLP : Merci, SQL. Je ne peux pas croire que ce soit le grand secret…

SQL : Ouais. Merci de m'avoir fait perdre mon temps.

ATELIER CONSTRUCTION DE SOUS-REQUÊTES

Parcourez chacun des scénarios ci-dessous. Suivez les instructions pour écrire les deux requêtes comme demandé, puis combinez-les dans une sous-requête.

1. Greg veut voir quel est le salaire moyen pour un développeur web dans sa table `emploi_actuel`. Puis il veut regarder ce que gagnent vraiment les gens par rapport au salaire moyen pour ce travail. S'il trouve des gens gagnant moins, il peut utiliser cela pour les cibler parce qu'ils pourraient être plus intéressés par un nouvel emploi.

   Écrivez une requête pour trouver le salaire moyen d'un développeur web de la table `emploi_actuel`.

2. Greg a besoin du prénom, du nom et du salaire de tous les développeurs web de sa table `emploi_actuel`.

   Écrivez une requête pour trouver le prénom, le nom et le salaire de tous les développeurs web de la table `emploi_actuel`.

3. Greg utilise le salaire moyen (et un peu de maths) comme sous-requête pour afficher chaque développeur web et combien ils gagnent en moins ou en plus du salaire moyen.

   Combinez les deux requêtes. Utilisez la sous-requête dans la liste des colonnes du `SELECT`.

ATELIER DE CONSTRUCTION DE SOUS-REQUÊTES – SOLUTION

Parcourez chacun des scénarios ci-dessous. Suivez les instructions pour écrire les deux requêtes comme demandé, puis combinez-les dans une sous-requête.

1. Greg veut voir quel est le salaire moyen pour un développeur web dans sa table `emploi_actuel`. Puis il veut regarder ce que gagnent vraiment les gens par rapport au salaire moyen pour ce travail. S'il trouve des gens gagnant moins, il peut utiliser cela pour les cibler parce qu'ils pourraient être plus intéressés par un nouvel emploi.

   Écrivez une requête pour trouver le salaire moyen d'un développeur web de la table `emploi_actuel`.

```
SELECT AVG(salaire) FROM emploi_actuel WHERE profession = 'Développeur web';
```

Le mot-clé AVG est exactement ce qu'il nous faut ici.

2. Greg a besoin du prénom, nom et du salaire de tous les développeurs web de sa table `emploi_actuel`.

   Écrivez une requête pour pour trouver le prénom, nom et salaire de tous les développeurs web de la table `emploi_actuel`.

```
SELECT mc.prenom, mc.nom, ea.salaire
FROM mes_contacts mc NATURAL JOIN emploi_actuel ea
WHERE ea.profession = 'Développeur web';
```

3. Greg utilise le salaire moyen (et un peu de maths) comme sous-requête pour afficher chaque développeur web et combien ils gagnent en moins ou en plus du salaire moyen.

   Combinez les deux requêtes. Utilisez la sous-requête dans la liste des colonnes du `SELECT`.

Voici notre sous-requête.

```
SELECT mc.prenom, mc.nom, ea.salaire,
ea.salaire – (SELECT AVG(salaire) FROM emploi_actuel WHERE profession = 'Développeur web')
FROM mes_contacts mc NATURAL JOIN emploi_actuel ea
WHERE ea.profession = 'Développeur web';
```

Une requête non-corrélée avec plusieurs valeurs : IN, NOT IN

Prenez la première requête que Greg a essayé à la page 403. Cela l'aide à repérer les gens avec des noms d'emplois qui ***correspondent*** à ses annonces. Cela prend l'ensemble complet de `professions` renvoyés par le `SELECT` dans la sous-requête et le compare à chaque ligne de la table `emploi_actuel` pour trouver les correspondances. possibles.

```
SELECT mc.prenom, mc.nom, mc.telephone, ea.profession

FROM emploi_actuel AS ea NATURAL JOIN mes_contacts AS mc

WHERE ea.profession IN (SELECT profession FROM offres_emplois);
```

IN compare chaque ligne des valeurs de ea.profession à l'ensemble renvoyé par la sous-requête.

Utiliser **`NOT IN`** peut aider Greg à voir les emplois qui ***ne correspondent pas*** à ses annonces. Cela prend l'ensemble des `professions` renvoyés par le `SELECT` de la sous-requête pour le comparer à chaque ligne de la table `emploi_actuel`, renvoyant toutes les valeurs qui *ne correspondent pas à* celles de la table `emploi_actuel`. Maintenant Greg peut essayer de trouver plus d'offres d'emplois pour ces professions.

```
SELECT mc.prenom, mc.nom, mc.telephone, ea.profession

FROM emploi_actuel ea NATURAL JOIN mes_contacts mc

WHERE ea.profession NOT IN (SELECT profession FROM offres_emplois);
```

NOT IN renvoie toutes les professions qui ne se trouvent pas dans les offres d'emplois.

Ces types de requêtes sont appelées **sous-requêtes non-corrélées**, quand `IN` ou `NOT IN` comparent les résultats de la sous-requête à ceux de la requête externe pour voir s'ils correspondent ou pas.

MUSCLEZ VOS NEURONES

Pourquoi ne pas simplement taper la liste des valeurs plutôt que d'utiliser une sous-requête ?

Une sous-requête non-corrélée utilise IN ou NOT IN pour tester si les valeurs renvoyées dans la sous-requête sont membres d'un ensemble (ou pas).

Écrivez des requêtes avec des jointures et des sous-requêtes non-corrélées quand nécessaire, pour répondre aux questions ci-dessous. Utilisez le schéma de la base de données `liste_greg` pour vous aider.

Plusieurs de ces requêtes nécessitent les fonctions d'agrégation que vous avez apprises au cours du problème de vente de cookies des Jeannettes.

Listez les professions dont les salaires égalent le salaire le plus élevé de la table offres_emplois.

→ Réponses page 406.

Listez les prénoms et noms des personnes avec un salaire supérieur au salaire moyen.

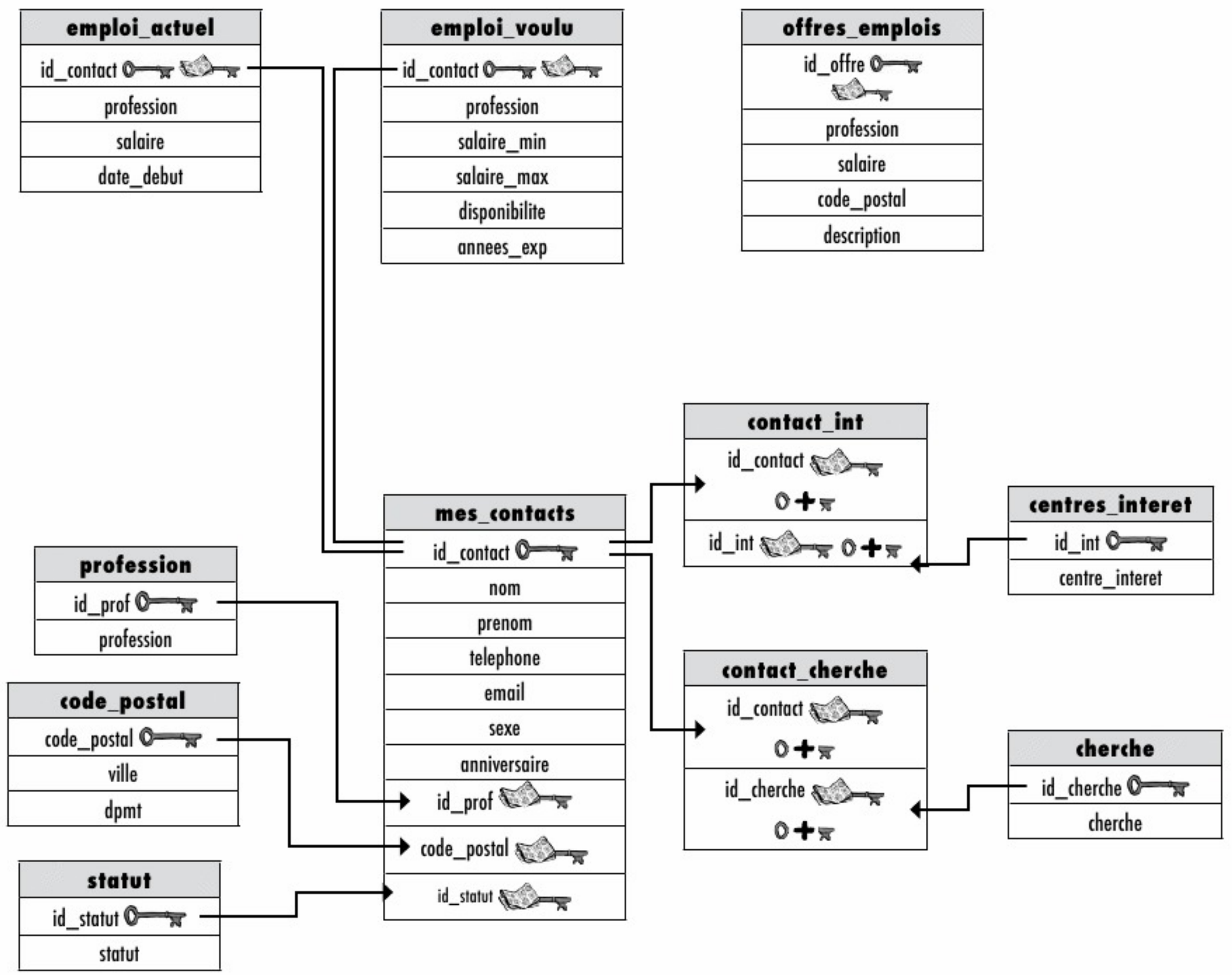
→ Réponses page 406.

Trouvez tous les concepteurs web qui ont le même code postal qu'une des offres d'emploi de concepteur web.

→ Réponses page 407.

Listez toutes les personnes qui ont le même code postal que la personne avec le plus haut salaire actuel.

→ Réponses page 407.


emploi_actuel
id_contact
profession
salaire
date_debut
emploi_voulu
id_contact
profession
salaire_min
salaire_max
disponibilite
annees_exp
offres_emplois
id_offre
profession
salaire
code_postal
description
contact_int
id_contact
id_int
centres_interet
id_int
centre_interet
mes_contacts
id_contact
nom
prenom
telephone
email
sexe
anniversaire
id_prof
code_postal
id_statut
profession
id_prof
profession
code_postal
code_postal
ville
dpmt
statut
id_statut
statut
contact_cherche
id_contact
id_cherche
cherche
id_cherche
cherche

Solutions des exercices

Écrivez des requêtes avec des jointures et des sous-requêtes non-corrélées quand nécessaire, pour répondre aux questions ci-dessous. Utilisez le schéma de la base de données `liste_greg` pour vous aider.

Listez les professions dont les salaires égalent le salaire le plus élevé de la table offres_emplois.

La requête externe cherche les correspondances avec la valeur MAX de salaire.

```
SELECT profession FROM offres_emplois
WHERE salaire = (SELECT MAX(salaire)
FROM offres_emplois);
```

La sous-requête renvoie une valeur unique.

MAX renvoie le plus gros salaire de la table.

Listez les prénoms et noms des personnes avec un salaire supérieur au salaire moyen.

La requête externe prend le résultat de la sous-requête et renvoie les correspondances qui sont supérieures.

```
SELECT mc.prenom, mc.nom
FROM mes_contacts mc
NATURAL JOIN emploi_actuel ea
WHERE ea.salaire > (SELECT AVG(salaire) FROM emploi_actuel);
```

La jointure naturelle nous donne les noms des gens avec un salaire supérieur à celui renvoyé par la requête interne.

La sous-requête renvoie le salaire moyen.

Trouvez tous les concepteurs web qui ont le même code postal qu'une des offres d'emploi de concepteur web.

Nous devons utiliser une jointure naturelle pour obtenir des infos utiles, comme les noms et les numéros de téléphone des gens que nous trouvons.

```
SELECT mc.prenom, mc.nom, mc.telephone FROM mes_contacts mc
NATURAL JOIN emploi_actuel ea WHERE ea.profession = 'concepteur web' AND mc.code_postal
IN (SELECT code_postal FROM offres_emplois WHERE profession = 'concepteur web');
```

Parce qu'il pourrait y avoir plus d'un code postal renvoyé, nous traitons les résultats comme un ensemble et utilisons "IN" pour trouver les correspondances.

La requête interne renvoie tous les codes postaux pour les offres d'emplois de concepteur web.

Listez toutes les personnes qui ont le même code postal que la personne avec le plus haut salaire actuel.

C'est une question piège, parce qu'il peut y avoir plusieurs personnes avec le plus haut salaire. Ce qui signifie que nous devons utiliser un IN. Nous avons aussi besoin de deux sous-requêtes.

La requête externe prend les codes postaux et trouve des correspondances dans la table mes_contacts. Puisque la sous-requête du milieu peut renvoyer plusieurs codes postaux, nous utilisons un IN.

La sous-requête du milieu trouve les codes postaux des gens qui ont le salaire maximum.

```
SELECT nom, prenom FROM mes_contacts
WHERE code_postal IN (SELECT mc.code_postal FROM mes_contacts mc
NATURAL JOIN emploi_actuel ea
WHERE ea.salaire = (SELECT MAX(salaire) FROM emploi_actuel));
```

La sous-requête la plus interne récupère le salaire MAX de la table emploi_actuel. Ce sera une valeur unique, nous pouvons donc utiliser =.

Sous-requêtes corrélées

Si une sous-requête non-corrélée signifie que la sous-requête est indépendante, alors je parie qu'une sous-requête corrélée est d'une certaine façon dépendante de la requête externe.

Exact. Dans une sous-requête non-corrélée, la requête interne, ou sous-requête, est interprétée par le SGBDR, suivie par la requête externe.

Ce qui nous laisse la sous-requête corrélée. Une sous-requête corrélée signifie que la requête interne s'appuie sur la requête externe pour être résolue.

La requête ci-dessous compte le nombre de centres d'intérêt de la table `centre_interet` pour chaque personne de `mes_contacts`, puis renvoie le prénom et le nom des personnes ayant trois centres d'intérêt.

```
SELECT mc.prenom, mc.nom
FROM mes_contacts AS mc
WHERE
3 = (
SELECT COUNT(*) FROM contact_int
WHERE id_contact = mc.id_contact
);
```

L'alias de mes_contacts est créé dans la requête externe.

La requête externe doit être exécutée avant que nous sachions quelle est la valeur de mc.id_contact.

La sous-requête fait référence à l'alias mc.

La sous-requête dépend de la requête externe. On a besoin de la valeur pour `id_contact` donnée par la requête externe avant que la requête interne puisse être traitée.

Elle utilise le **même alias** ou **nom de corrélation** pour `mes_contacts`, `mc`, que celui créé dans la requête externe.

Une requête corrélée (utile) avec NOT EXISTS

Une utilisation très courante des sous-requêtes corrélées est de trouver toutes les lignes de la requête externe pour lesquelles il n'existe aucune ligne dans une table liée.

Supposez que Greg ait besoin de plus de clients pour son entreprise de recrutement en expansion et veuille envoyer un email à toutes les personnes de `mes_contacts` qui **ne sont pas** actuellement dans la table **`emploi_actuel`** table. Il peut utiliser `NOT EXISTS` pour cibler ces personnes.

```
SELECT mc.prenom prenom, mc.nom nom, mc.email email
FROM mes_contacts mc
WHERE NOT EXISTS
(SELECT * FROM emploi_actuel ea
WHERE mc.id_contact = ea.id_contact );
```

NOT EXISTS trouve les prénoms, noms et adresses email des personnes de la table mes_contacts qui ne sont actuellement pas listées dans la table emploi_actuel.

Quel type de données ?

Reliez chaque partie de la requête ci-dessous à ce qu'elle fait.

`mc.prenom prenom`	Définit un alias pour le champ mc.nom
`WHERE NOT EXISTS`	Si deux id_contact sont égaux, une condition est remplie
`WHERE mc.id_contact = ea.id_contact`	Définit un champ avec « prenom » comme alias
`FROM mes_contacts mc`	Sélectionne tous les champs de la table dont l'alias est « ea »
`mc.nom nom`	Définit un champ pour qu'il ait « email » comme alias
`SELECT * FROM emploi_actuel ea`	Spécifie l'action si quelque chose n'est pas trouvé
`mc.email email`	Définit un alias pour mes_contacts

EXISTS et NOT EXISTS

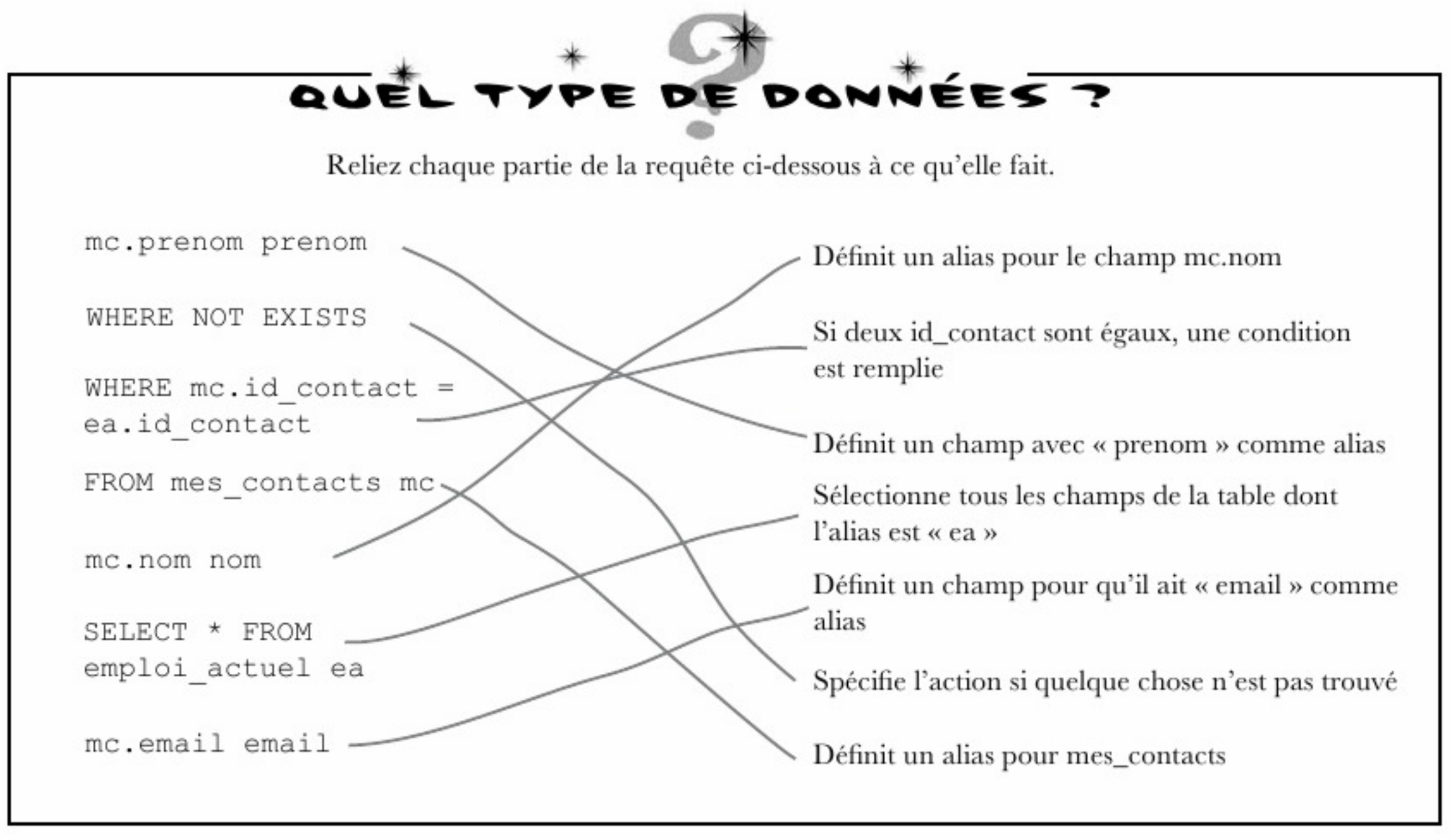
Tout comme avec `IN` et `NOT IN`, vous pouvez utiliser **`EXISTS`** et `NOT EXISTS` avec vos sous-requêtes. La requête ci-dessous renvoie les données de `mes_contacts` si `id_contact` apparaît au moins une fois dans la table `contact_int`.

```
SELECT mc.prenom prenom, mc.nom nom, mc.email email
FROM mes_contacts mc
WHERE EXISTS
(SELECT * FROM contact_int ci WHERE mc.id_contact = ci.id_contact );
```

EXISTS trouve les prénoms, noms et adresses email des gens de la table mes_contacts dont l'id_contact apparaît au moins une fois dans la table contact_int.

QUEL TYPE DE DONNÉES ?

Reliez chaque partie de la requête ci-dessous à ce qu'elle fait.



À vos crayons !

Écrivez une requête qui renvoie l'email des gens qui ont au moins un centre d'intérêt mais n'existent pas dans la table emploi_actuel.

→ Réponses page 416.

Le Service de recrutement de Greg est ouvert

Greg n'a plus aucun problème pour récupérer ses données avec des sous-requêtes. Il découvre même qu'il peut les utiliser dans les instructions `INSERT`, `UPDATE` et `DELETE`.

Il loue un petit bureau pour sa nouvelle entreprise et décide de faire une grande fête de lancement.

il n'y a pas de

Questions stupides

Q: Alors peut-on mettre une sous-requête dans une sous-requête ?

R: Certainement. Il y a une limite au nombre de sous-requêtes gigognes que vous pouvez utiliser, mais la plupart des systèmes SGBDR vous en permettent bien plus que ce que vous ne pourriez facilement utiliser.

Q: Quelle est la meilleure approche quand on essaie de construire une sous-requête à l'intérieur d'une sous-requête ?

R: Votre meilleure option est d'écrire de petites requêtes pour les différentes parties de la question. Puis regardez-les et voyez comment vous devez les combiner. Si vous essayez de trouver des gens qui gagnent autant d'argent que le concepteur web le mieux payé, séparez cela en :

Trouver le salaire du concepteur web le mieux payé
Trouver les gens qui gagnent x comme salaire

puis placez la première réponse à la place du **x**.

Q: Si je n'aime pas utiliser des sous-requêtes, y a-t-il moyen d'utiliser des jointures à la place ?

R: La plupart du temps, oui. Vous devez d'abord découvrir quelques autres jointures, cependant. Ce qui nous amène à...

En route pour la fête

Greg repère cette inquiétante couverture de magazine :

HEBDO

ENQUÊTEUR

La VÉRITE CHOC sur les sous-requêtes RÉVELÉE!

DES JOINTURES DÉGUISÉES

Des voisins disent que les sous-requêtes ne peuvent « rien faire de plus » que des jointures et que « la vérité doit enfin être révélée ».

Par Fleur Delacour
JOURNALISTE À L'ENQUÊTEUR

DATAVILLE – Ce qui, pendant de nombreuses années, n'a été que spéculations a maintenant été vérifié par les sources de l'Enquêteur. Les jointures et les sous-requêtes peuvent être utilisées pour faire exactement les mêmes requêtes. À la grande confusion des habitants, tout ce que l'on peut faire avec une sous-requête peut être fait avec certains types de jointures.

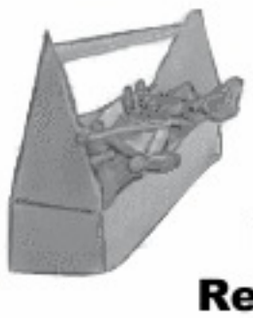
« C'est terrible », sanglotte l'institutrice Charlotte Choucroute. « Comment puis-je dire aux enfants que ce qu'ils pensaient savoir sur les sous-requêtes, toutes ces heures passées à apprendre à les utiliser, hé bien, ils auraient pu simplement utiliser des jointures. Ça me fend le cœur. »

On peut s'attendre à ce que les retombées de cette révélation continuent loin dans le prochain chapitre, alors que les jointures externes seront exposées à l'examen du public.

Charlotte Choucroute, une habitante de la ville, a été choquée d'apprendre la vérité sur les sous-requêtes.

ETAIT-CE UNE PERTE DE TEMPS ? LES SOUS-REQUÊTES SONT-ELLES VRAIMENT LA MÊME CHOSE QUE DES JOINTURES ?

RENDEZ-VOUS AU PROCHAIN CHAPITRE POUR LE SAVOIR.



Boîte à outils SQL

Vous avez terminé le chapitre 9 et maîtrisez l'art de la sous-requête. Regardez tout ce que vous avez appris. Pour une liste exhaustive des outils du livre, voyez l'annexe C.

Requête externe

Une requête qui contient une requête interne ou sous-requête.

Requête interne

Une requête dans une autre requête. Également appelée une sous-requête.

Sous-requête non-corrélée

Une sous-requête qui est indépendante et ne fait aucune référence à la requête externe.

Sous-requête corrélée

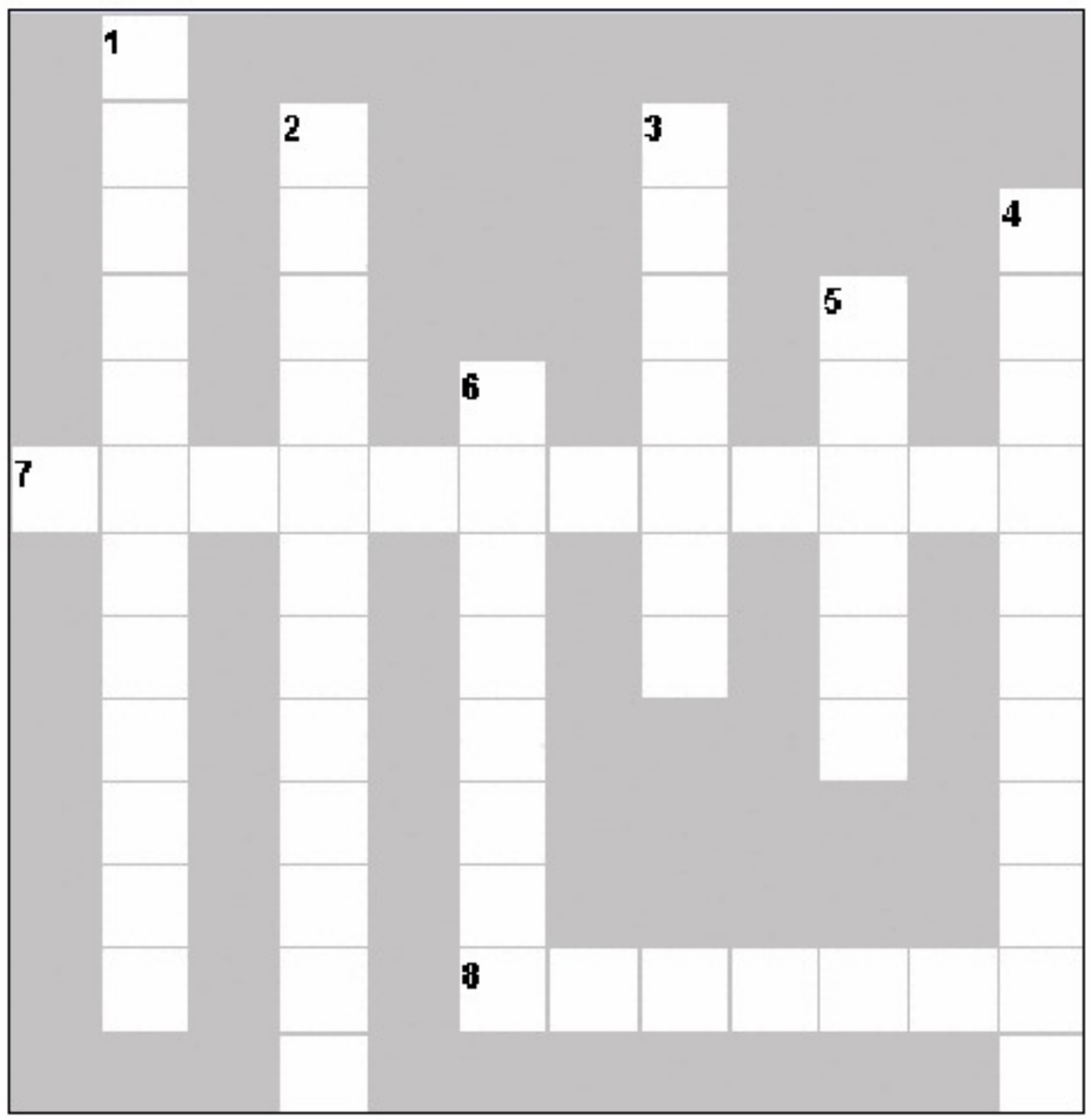
Une sous-requête qui s'appuie sur des valeurs renvoyées par la requête externe.

Sous-requête

Une requête enveloppée dans une autre requête. Également appelée une requête interne.

Sous-requêtes-croisées

Vous pouvez différencier votre requête interne de votre requête externe, mais pouvez-vous résoudre ces mots-croisés ? Toutes les solutions se trouvent dans ce chapitre.



Définitions horizontales :

7. Si la requête est indépendante et ne fait pas référence à la requête externe, c'est une requête ... .

8. La requête ... contient la requête interne ou sous-requête.

Définitions verticales :

1. Dans une sous-requête ..., la requête interne ou sous-requête, est interprétée par le SGBDR, suivie par la requête externe.

2. Une requête à l'intérieur d'une autre requête s'appelle une ... .

3. La requête ... s'appelle la sous-requête.

4. Les sous-requêtes sont toujours à l'intérieur de ... .

5. Une sous-requête est toujours une unique instruction ... .

6. Une sous-requête ... signifie que la requête interne s'appuie sur la requête externe pour être résolue.

À vos crayons ! Solution

De la page 411.

Écrivez une requête qui renvoie l'email des gens qui ont au moins un centre d'intérêt mais n'existent pas dans la table emploi_actuel.

```
SELECT mc.email FROM mes_contacts mc WHERE
EXISTS
(SELECT * FROM contact_int ci WHERE mc.id_contact = ci.id_contact)
AND
NOT EXISTS
(SELECT * FROM emploi_actuel ea
WHERE mc.id_contact = ea.id_contact );
```

Comme pour n'importe quels deux éléments qui doivent être vrais tous les deux, vous pouvez utiliser AND dans votre clause WHERE.

Sous-requêtes-croisées - Solution

	1 N										
	O		2 S				3 I				
	N		O				N				4 P
	-		U				T		5 S		A
	C		S		6 C		E		E		R
7 N	O	N	-	C	O	R	R	É	L	É	E
	R		R		R		N		E		N
	R		E		R		E		C		T
	É		Q		É				T		H
	L		U		L						È
	É		Ê		É						S
	E		T		8 E	X	T	E	R	N	E
			E								S

Nouvelles tactiques

Nettoyage des vieilles données

Je voudrais nettoyer ma table profession. Je pense qu'il s'y trouve des valeurs que je n'utilise plus. Comment puis-je trouver facilement les professions qui ne sont reliées à aucun des enregistrements de la table mes_contacts ? Je n'y arrive pas avec une jointure interne.

Vous pouvez obtenir ces informations avec une jointure externe.

Regardons ce que font les jointures externes, puis nous vous montrerons comment trouver ces professions que vous n'utilisez plus.

Une jointure externe renvoie toutes les lignes d'une des tables, ainsi que les informations correspondantes d'une autre table.

Avec une jointure interne, vous ***comparez les lignes de deux tables***, mais l'**ordre de ces deux tables n'a pas d'importance**.

Rappelons-nous brièvement ce que fait une équi-jointure. Nous obtenons toutes les colonnes qui correspondent à `id_jouet` dans les deux tables. Elle fait correspondre l'`id_jouet` qui existe dans les deux tables :

```
SELECT f.fille, j.jouet
FROM filles f
INNER JOIN jouets j
ON f.id_jouet = j.id_jouet;
```

L'équi-jointure compare les lignes de ces deux tables pour obtenir le résultat. Elle relie les valeurs id.

filles

id_fille	fille	id_jouet
1	Jeanne	3
2	Sarah	4
3	Sandrine	1

jouets

id_jouet	jouet
1	cerceau
2	maquette d'avion
3	petits soldats
4	harmonica
5	jeu de cartes
6	mécano
7	ardoise magique
8	ressort fluo

Nos résultats

fille	jouet
Sandrine	cerceau
Jeanne	petits soldats
Sarah	harmonica

À propos de gauche et droite

Par comparaison, les jointures externes concernent plus la ***relation entre deux tables*** que les jointures que vous avez vues jusqu'ici.

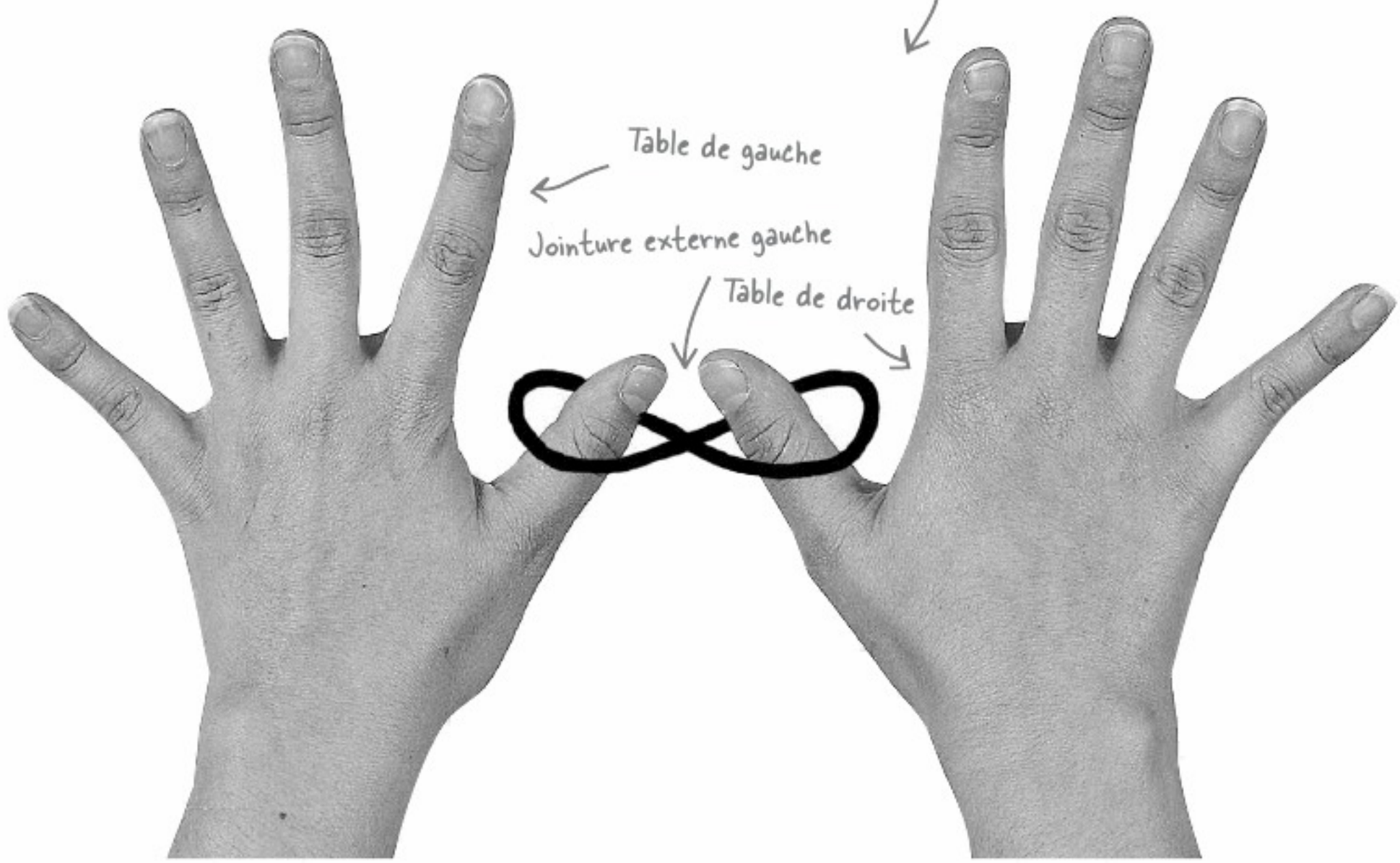
Une `JOINTURE EXTERNE` **`GAUCHE`** prend ***toutes les lignes*** de la **table de gauche** et **les fait correspondre aux lignes** de la table de **`DROITE`**. C'est utile quand la table de gauche et celle de droite ont une relation un-à-plusieurs.

La jointure externe gauche fait correspondre TOUTES LES LIGNES de la table de GAUCHE avec une ligne de la table de droite.

Le grand secret pour comprendre une jointure externe est de savoir quelle table est à gauche et laquelle est à droite.

Dans une **`JOINTURE EXTERNE GAUCHE`**, la table qui vient après `FROM` et **AVANT** la jointure est la table de **GAUCHE** et celle qui vient **APRÈS** la jointure est la **table de DROITE.**

La table qui vient avant la jointure que vous utilisez (gauche ou droite) est la table de gauche.



Voici une jointure externe gauche

Nous pouvons utiliser une jointure externe gauche pour trouver quelle fille a quel jouet.

Voici la syntaxe d'une jointure externe gauche utilisant les mêmes tables qu'avant. La table `filles` est la première après `FROM`, c'est donc la table de `GAUCHE`, puis nous avons le `LEFT OUTER JOIN`, et enfin, la table `jouets` est la table de `DROITE` :

Alors, le LEFT OUTER JOIN prend toutes les lignes de la table de gauche (la table filles) et les fait correspondre à des lignes de la table de DROITE (la table jouets).

```
SELECT f.fille, j.jouet
FROM filles f
LEFT OUTER JOIN jouets j
ON f.id_jouet = j.id_jouet;
```

Elle vient avant la jointure externe gauche, donc filles est la table de gauche...

... et parce qu'elle vient après la jointure externe gauche, jouets est la table de droite.

Elle vient avant la jointure externe gauche, filles est donc la table de gauche...

... et parce qu'elle vient après la jointure externe gauche, jouets est la table de droite.

filles

id_fille	fille	id_jouet
1	Jeanne	3
2	Sarah	4
3	Sandrine	1

jouets

id_jouet	jouet
1	cerceau
2	maquette d'avion
3	petits soldats
4	harmonica
5	jeu de cartes
6	mécano
7	ardoise magique
8	ressort fluo

Les résultats de la jointure externe gauche

Nos résultats sont les mêmes que ceux de la jointure interne.

Nos résultats.

fille	jouet
Sandrine	cerceau
Jeanne	petits soldats
Sarah	harmonica

Qu'est-ce que c'est que ça ? Pourquoi en faire tout un plat alors ? On dirait qu'une jointure externe est la même chose qu'une jointure interne.

La différence est qu'une jointure externe vous donne une ligne, qu'il y ait une correspondance dans l'autre table ou pas.

Et une valeur NULL vous dit qu'il n'y a pas de correspondance. Dans le cas de nos filles et jouets, une valeur NULL dans les résultats signifie qu'une des filles n'a aucun jouet. C'est une information précieuse !

Une valeur NULL dans les résultats d'une jointure externe gauche signifie que la table de droite n'a aucune valeur qui corresponde à la table de gauche.

À vos crayons !

Voici une requête où nous avons inversé l'ordre de nos tables. Dessinez la table des résultats de cette requête, telle qu'elle sera selon vous.

```
SELECT f.fille, j.jouet
FROM jouets j
LEFT OUTER JOIN filles f
ON f.id_jouet = j.id_jouet;
```

(Indice: il y aura 8 lignes dans la table de résultats.)

À vos crayons ! Solution

Voici une requête où nous avons inversé l'ordre de nos tables. Dessinez la table des résultats de cette requête, telle qu'elle sera selon vous.

```
SELECT f.fille, j.jouet
FROM jouets j
LEFT OUTER JOIN filles f
ON f.id_jouet = j.id_jouet;
```

La table de gauche.

La table de droite.

Cette fois, toutes les lignes de la table jouets (la table de gauche) sont comparées à la table filles (la table de droite).

La table de gauche

La table de droite

jouets

id_jouet	jouet
1	cerceau
2	maquette d'avion
3	petits soldats
4	harmonica
5	jeu de cartes
6	mecano
7	ardoise magique
8	ressort fluo

filles

id_fille	fille	id_jouet
1	Jeanne	3
2	Sarah	4
3	Sandrine	1

Voici ce que nous obtenons avec l'ordre des tables modifié :

fille	jouet
Sandrine	cerceau
NULL	maquette d'avion
Jeanne	petits soldats
Sarah	harmonica
NULL	jeu de cartes
NULL	mecano
NULL	ardoise magique
NULL	ressort fluo

Si une correspondance est trouvée, elle apparaît comme résultat dans notre table. Si aucune correspondance n'est trouvée, nous obtenons toujours une ligne dans notre table, mais avec NULL pour la valeur sans correspondance.

L'ordre dans lequel les colonnes apparaissent dans la table est l'ordre dans lequel nous les SELECTionnons. Cet ordre n'a rien à voir avec la jointure GAUCHE.

Ci-dessous se trouvent deux ensembles de résultats. Pour chacun d'eux, écrivez une jointure externe gauche qui pourrait l'avoir créé, ainsi qu'une table filles et une table jouets avec les données qui correspondent aux résultats.

La requête

Résultat d'une jointure externe gauche :

fille	jouet
Juliette	pistolet à eau
Zoé	culbuto
Emma	NULL

Table de gauche

Nous avons fait celle-ci pour vous.

filles

id_fille	fille	id_jouet
1	Juliette	1
2	Zoé	2
3	Emma	3

Table de droite

La requête

Résultat d'une jointure externe gauche :

Celle-ci n'est pas évidente.

fille	jouet
Juliette	pistolet à eau
Zoé	pistolet à eau
NULL	culbuto
Sarah	ressort fluo
Mathilde	ressort fluo

Table de gauche

Table de droite

Ci-dessous se trouvent deux ensembles de résultats. Pour chacun d'eux, écrivez une jointure externe gauche qui pourrait l'avoir créé, ainsi qu'une table filles et une table jouets avec les données qui correspondent aux résultats.

La requête

```
SELECT f.fille, j.jouet
FROM filles f
LEFT OUTER JOIN jouets j
ON f.id_jouet = j.id_jouet;
```

Résultat d'une jointure externe gauche :

fille	jouet
Juliette	pistolet à eau
Zoé	culbuto
Emma	NULL

Voici les jouets apparaissant dans nos résultats.

Table de gauche

filles

id_fille	fille	id_jouet
1	Juliette	1
2	Zoé	2
3	Emma	3

Table de droite

jouets

id_jouet	jouet
1	pistolet à eau
2	culbuto

Ce peut être n'importe quel id_jouet qui n'existe pas dans la table jouets puisque la colonne jouet donne un NULL dans les résultats.

Les valeurs répétées signifient que plusieurs filles ont le même jouet.

La requête

```
SELECT f.fille, j.jouet
FROM jouets j
LEFT OUTER JOIN filles f
ON f.id_jouet = j.id_jouet;
```

Résultat d'une jointure externe gauche :

fille	jouet
Juliette	pistolet à eau
Zoé	pistolet à eau
NULL	culbuto
Sarah	ressort fluo
Mathilde	ressort fluo

Et le NULL signifie qu'aucune fille n'a de culbuto.

Table de gauche

jouets

id_jouet	jouet
1	pistolet à eau
2	culbuto
3	ressort fluo

Table de droite

filles

id_fille	fille	id_jouet
1	Juliette	1
2	Zoé	1
3	Sarah	3
4	Mathilde	3

Jointures externes et correspondances multiples

Comme vous venez de le remarquer dans l'exercice, vous obtenez des lignes même quand il n'y a aucune correspondance dans l'autre table et plusieurs lignes quand il y a plusieurs correspondances. Voici ce que fait concrètement la jointure externe gauche :

```
SELECT f.fille, j.jouet
FROM jouets j
LEFT OUTER JOIN filles f
ON f.id_jouet = j.id_jouet;
```

jouets

id_jouet	jouet
1	pistolet à eau
2	culbuto
3	ressort fluo

filles

id_fille	fille	id_jouet
1	Juliette	1
2	Zoé	1
3	Sarah	3
4	Mathilde	3

La ligne pistolet à eau de `jouets` est comparée à la ligne Juliette de `filles` : jouets.id_jouet = 1, filles.id_jouet = 1
Nous avons une correspondance.
La ligne pistolet à eau de `jouets` est comparée à la ligne Zoé de `filles` : jouets.id_jouet = 1, filles.id_jouet = 1
Nous avons une correspondance.
La ligne pistolet à eau de `jouets` est comparée à la ligne Sarah de `filles` : jouets.id_jouet = 1, filles.id_jouet = 3
Pas de correspondance.
La ligne pistolet à eau de `jouets` est comparée à la ligne Mathilde de `filles` : jouets.id_jouet = 1, filles.id_jouet = 3
Pas de correspondance.
La ligne culbuto de `jouets` est comparée à la ligne Juliette de `filles` : jouets.id_jouet = 2, filles.id_jouet = 1
Pas de correspondance.
La ligne culbuto de `jouets` est comparée à la ligne Zoé de `filles` : jouets.id_jouet = 2, filles.id_jouet = 1
Pas de correspondance.
La ligne culbuto de `jouets` est comparée à la ligne Sarah de `filles` : jouets.id_jouet = 2, filles.id_jouet = 3
Pas de correspondance.
La ligne culbuto de `jouets` est comparée à la ligne Mathilde de `filles` : jouets.id_jouet = 2, filles.id_jouet = 3
Pas de correspondance.
Fin de table, une ligne est créée avec un NULL.
La ligne ressort fluo de `jouets` est comparée à la ligne Juliette de `filles` : jouets.id_jouet = 3, filles.id_jouet = 1
Pas de correspondance.
La ligne ressort fluo de `jouets` est comparée à la ligne Zoé de `filles` : jouets.id_jouet = 3, filles.id_jouet = 1
Pas de correspondance.
La ligne ressort fluo de `jouets` est comparée à la ligne Sarah de `filles` : jouets.id_jouet = 3, filles.id_jouet = 3
Nous avons une correspondance.
La ligne ressort fluo de `jouets` est comparée à la ligne Mathilde de `filles` : jouets.id_jouet = 3, filles.id_jouet = 3
Nous avons une correspondance.

fille	jouet
Juliette	pistolet à eau
Zoé	pistolet à eau
NULL	culbuto
Sarah	ressort fluo
Mathilde	ressort fluo

La jointure externe droite

La jointure externe droite est exactement la même chose que la jointure externe gauche, sauf qu'elle compare la table de droite à celle de gauche. Les deux requête ci-dessous vous donnent exactement les mêmes résultats :

La jointure externe droite évalue la table de droite par rapport à celle de gauche.

```
SELECT f.fille, j.jouet
FROM jouets j
RIGHT OUTER JOIN filles f
ON f.id_jouet = j.id_jouet;
```

La table de gauche

La table de droite.

```
SELECT f.fille, j.jouet
FROM filles f
LEFT OUTER JOIN jouets j
ON f.id_jouet = j.id_jouet;
```

La table de gauche

La table de droite

Vous avez déjà vu cette requête page 420.

La table de droite dans la première requête et celle de gauche dans la deuxième.

Ces deux requêtes font de la table filles la table de droite ou la table de gauche mais donnent les mêmes résultats.

La table de gauche dans la première requête et celle de droite dans la deuxième.

filles

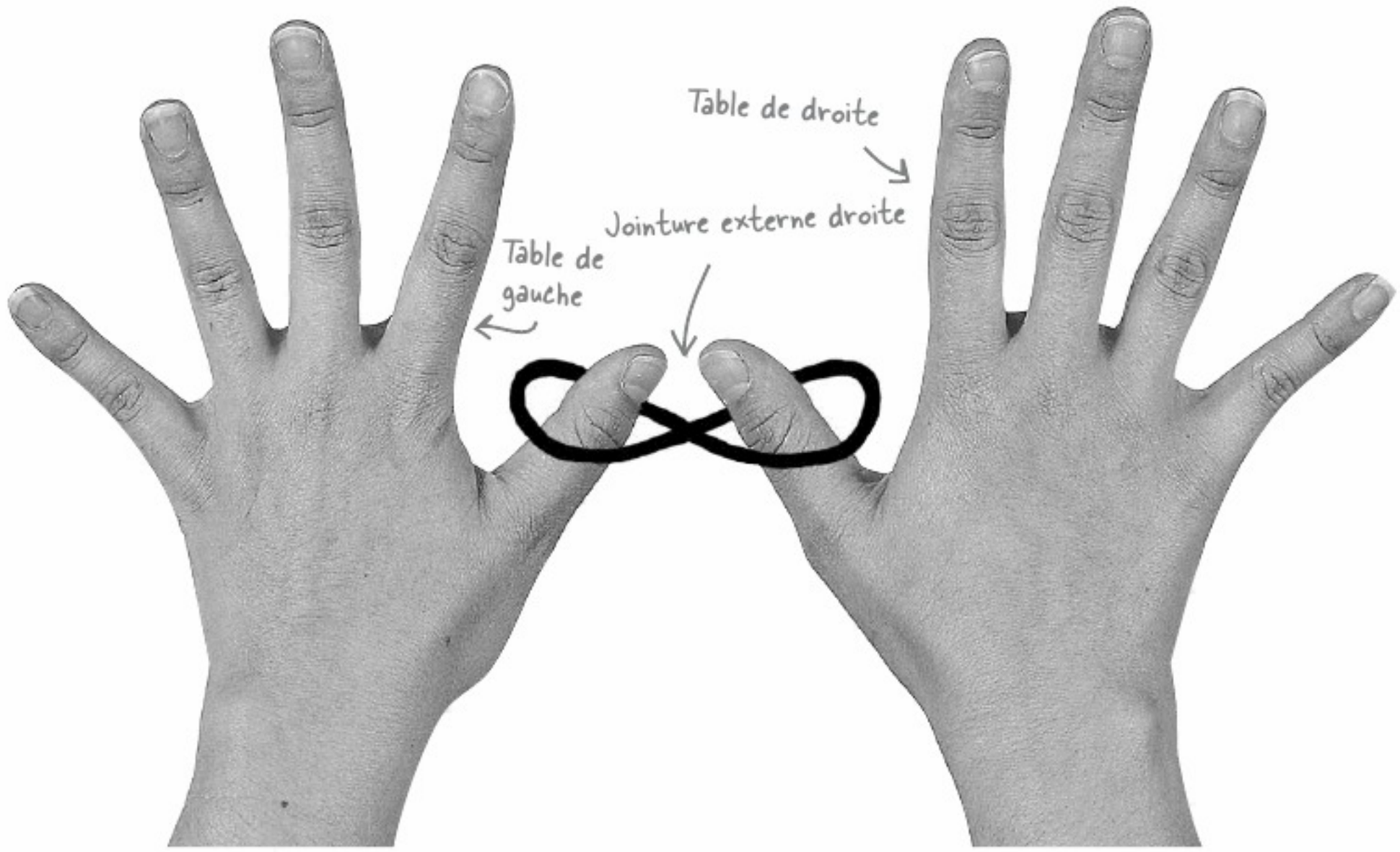
id_fille	fille	id_jouet
1	Jeanne	3
2	Sarah	4
3	Sandrine	1

jouets

id_jouet	jouet
1	cerceau
2	maquette d'avion
3	petits soldats
4	harmonica
5	jeu de cartes
6	mécano
7	ardoise magique
8	ressort fluo

Nos résultats

fille	jouet
Sandrine	cerceau
Jeanne	petits soldats
Sarah	harmonica



il n'y a pas de

Questions stupides

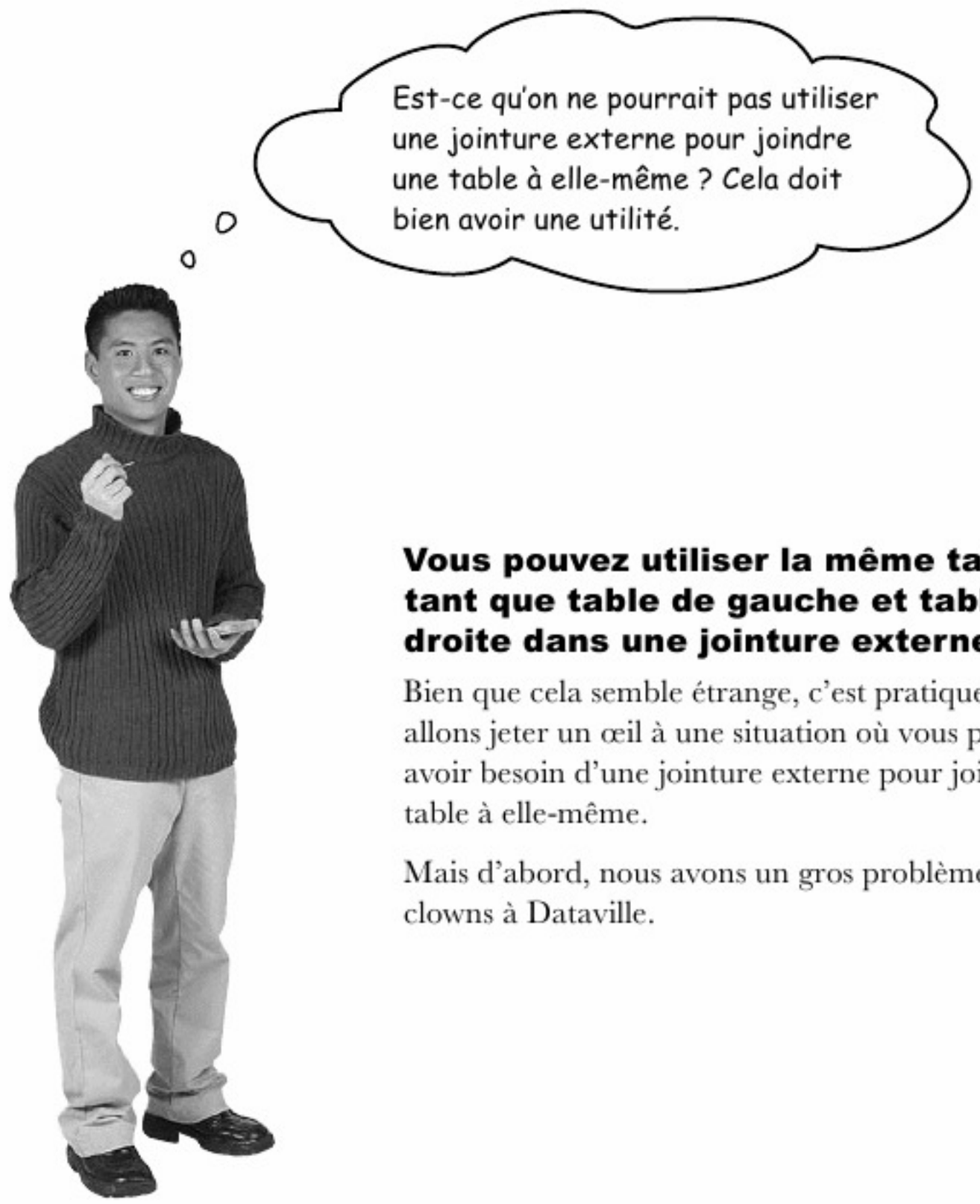
Q: Y a-t-il une raison d'utiliser une jointure externe gauche plutôt qu'une droite ?

R: Changer le mot LEFT en RIGHT est plus facile que de changer l'ordre des tables dans la requête. Vous n'avez qu'un mot à changer au lieu d'inverser les noms des tables et leurs alias.

Mais en général, il peut être plus facile d'en choisir un des deux, disons la jointure externe gauche, et de toujours l'utiliser, en changeant la table qui est à gauche et celle qui est à droite. Cela peut être plus clair.

Q: S'il y a une jointure externe GAUCHE et une jointure externe DROITE, existe-t-il une jointure qui renvoie à la fois les résultats de la gauche et de la droite ?

R: Elle existe dans certains SGBDR, mais pas dans tous. Elle s'appelle `FULL OUTER JOIN`, jointure externe complète. Mais elle ne fonctionne pas avec MySQL, SQL Server ou Access.



Vous pouvez utiliser la même table en tant que table de gauche et table de droite dans une jointure externe.

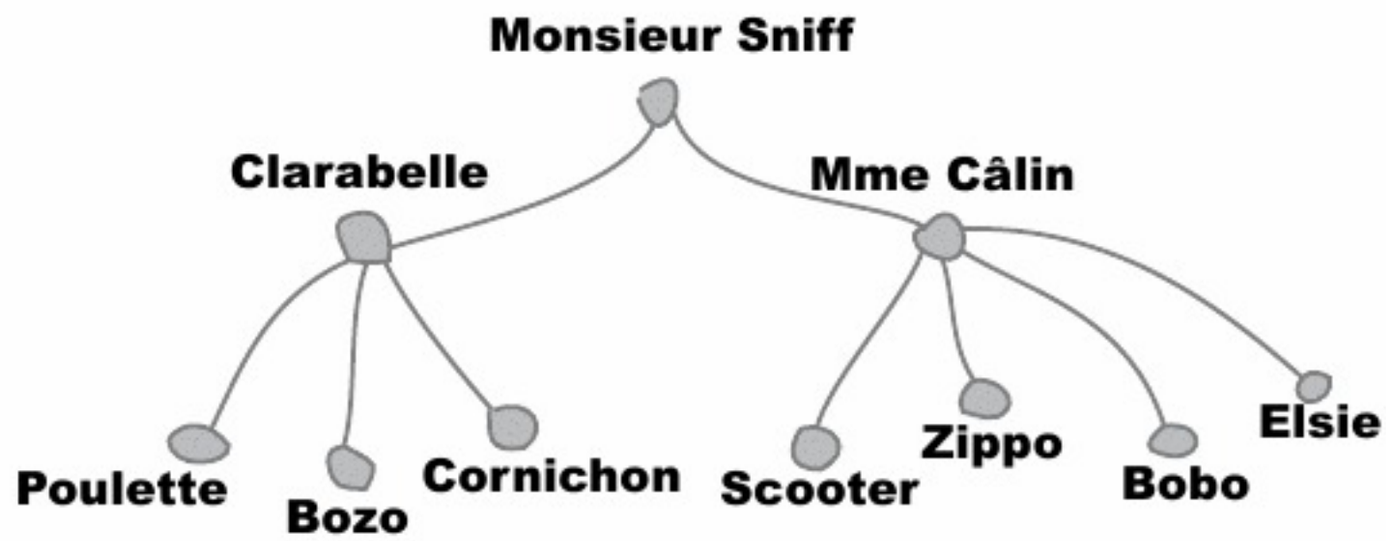
Bien que cela semble étrange, c'est pratique. Nous allons jeter un œil à une situation où vous pourriez avoir besoin d'une jointure externe pour joindre une table à elle-même.

Mais d'abord, nous avons un gros problème de clowns à Dataville.

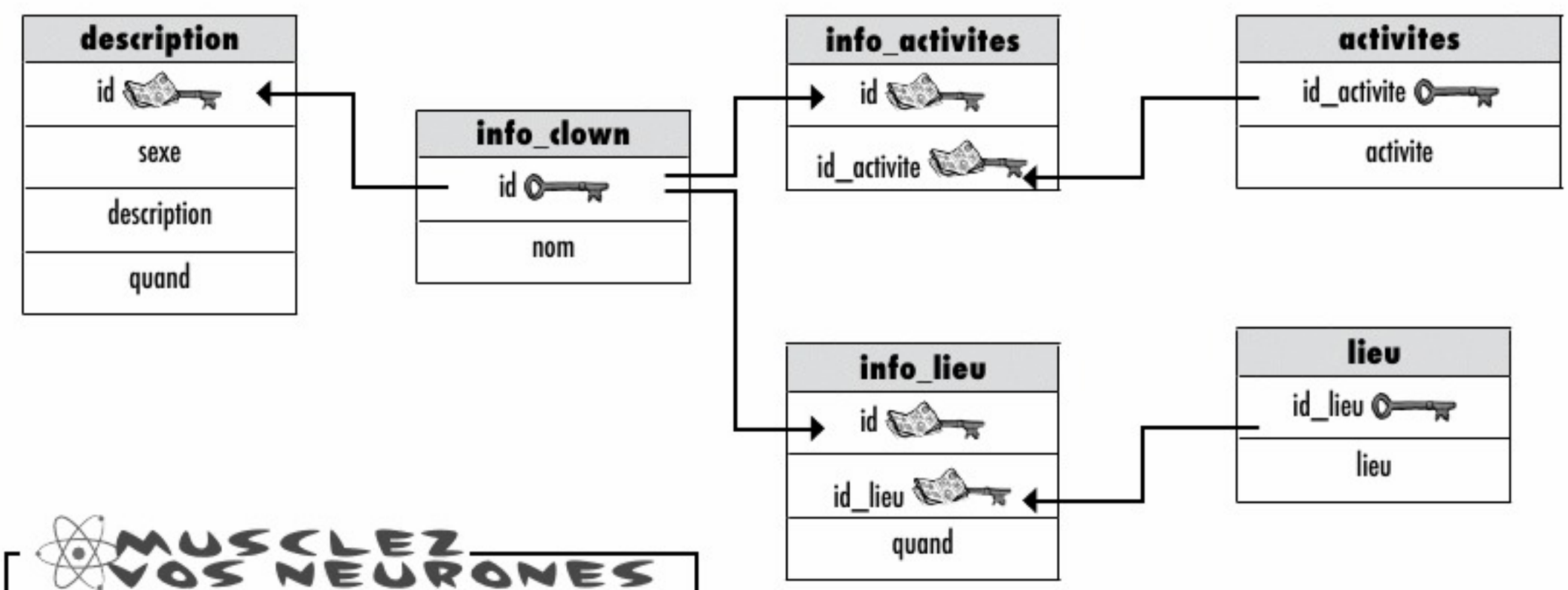
Pendant que vous faisiez des jointures externes...

De retour à Dataville, les clowns s'organisent et des chefs de clowns sont nommés responsables. C'est une évolution inquiétante et nous devons nous tenir au courant de qui sont ces chefs et quels clowns dépendent de quel chef.

Voici un exemple de la nouvelle hiérarchie des clowns. Chaque clown a un chef, sauf le clown à leur tête, Monsieur Sniff.



Regardons notre schéma actuel pour voir comment intégrer au mieux ces nouvelles informations :

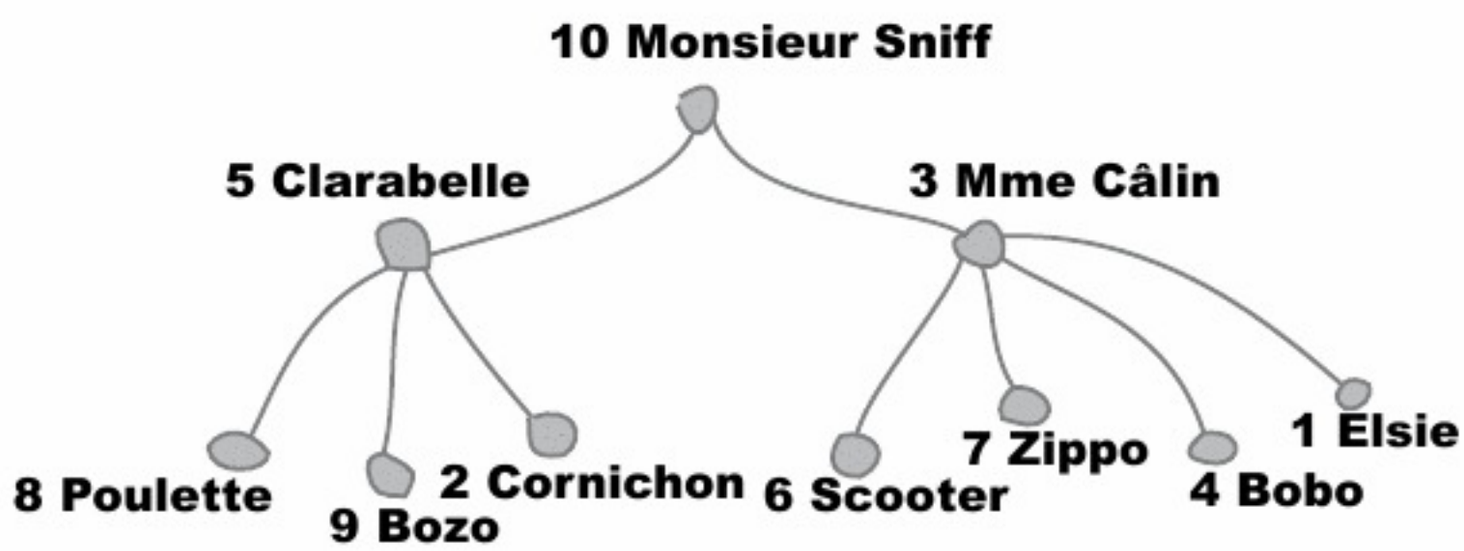


MUSCLEZ VOS NEURONES

Comment pouvez-vous restructurer votre schéma pour intégrer les informations sur les chefs des clowns ?

On peut créer une nouvelle table

Nous pouvons créer une table qui liste chaque clown et l'id de son chef. Voici notre hiérarchie avec les id de chaque clown.



Et voici une nouvelle table qui liste chaque clown et l'`id` de son chef dans la table `info_clown`.

`chefs_clown`

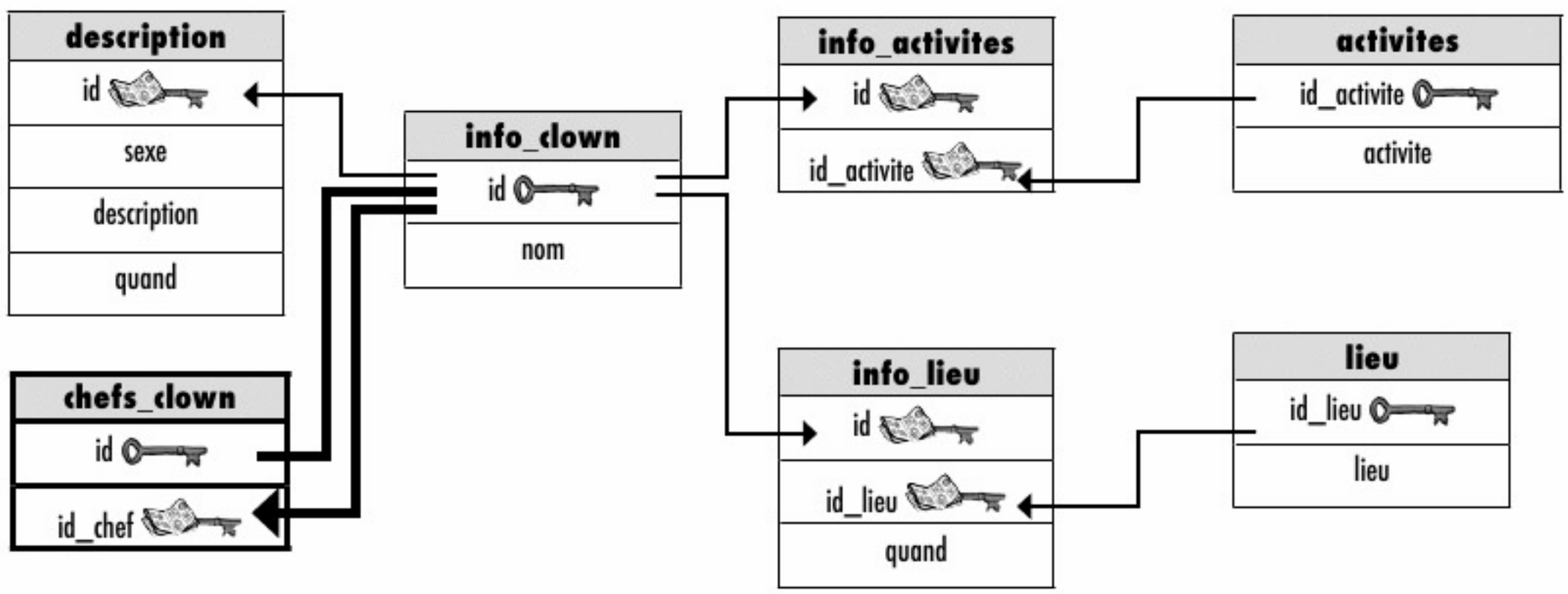
id	id_chef
1	3
2	5
3	10
4	3
5	10
6	3
7	3
8	5
9	5
10	10

Nous avons une relation un-à-un entre la table `chefs_clown` et la table `info_clown`.

Monsieur Sniff n'a pas de chef, mais il a besoin d'un id. Nous pouvons lui donner son propre id pour id_chef et éviter un NULL dans cette colonne.

Comment s'intègre la nouvelle table

Regardons notre schéma actuel pour voir comment intégrer au mieux cette nouvelle table :



C'est un peu bizzare. Nous avons une relation un-à-un avec `id`, notre clé primaire, et une relation un-à-plusieurs avec `id_chef`. Nous avons une clé primaire et une clé étrangère ***venant toutes les deux de la table info_clown***.

MUSCLEZ VOS NEURONES

Y a-t-il un moyen de conserver un suivi des chefs des clowns sans créer une toute nouvelle table ?

Une clé étrangère auto-référencée

Il nous faut une nouvelle colonne dans notre table `info_clown` pour nous dire qui est le chef de chaque clown. La nouvelle colonne contiendra le numéro id du chef de chaque clown. Nous l'appellerons `id_chef`, tout comme nous l'avions fait dans la table `chefs_clown`.

Dans la table `chefs_clown`, `id_chef` était une clé étrangère. Si nous ajoutons la colonne à `info_clown`, c'est toujours une clé étrangère, même si elle est dans la table `info_clown`. C'est ce qu'on appelle une **clé étrangère auto-référencée**. Auto-référencée signifie que c'est une clé qui fait référence à un autre champ dans la même table.

Nous assumons que Monsieur Sniff est son propre chef, son `id_chef` est donc le même que son `id`. Cela signifie que nous pouvons utiliser une clé étrangère auto-référencée pour notre `id_chef`.

Une **clé étrangère auto-référencée** est la ***clé primaire*** d'une table ***utilisée dans cette même table, dans un autre but***.

Une clé étrangère AUTO-RÉFÉRENCÉE est la clé primaire d'une table utilisée dans cette même table dans un autre but.

Voici la nouvelle colonne id_chef que nous avons simplement ajoutée à la table info_clown. Elle contient une clé étrangère auto-référencée.

info_clown

id	nom	id_chef
1	Elsie	3
2	Cornichon	5
3	Mme Câlin	10
4	Bobo	3
5	Clarabelle	10
6	Scooter	3
7	Zippo	3
8	Poulette	5
9	Bozo	5
10	Monsieur Sniff	10

Ceci référence le champ id de cette même table pour nous dire quel clown est le chef d'Elsie.

Encore une fois, l'id_chef de Monsieur Sniff est son propre id.

Joindre une table à elle-même

Supposez que nous voulions répertorier tous les clowns et leurs chefs. Nous pouvons facilement obtenir une liste des noms de chaque clown et de l'id de leur chef avec ce `SELECT` :

```
SELECT nom, id_chef FROM info_clown;
```

Mais ce que nous voulons en fait, c'est le nom du clown et celui de son chef :

nom	boss
Elsie	Mme Câlin
Cornichon	Clarabelle
Mme Câlin	Monsieur Sniff
Bobo	Mme Câlin
Clarabelle	Monsieur Sniff
Scooter	Mme Câlin
Zippo	Mme Câlin
Poulette	Clarabelle
Bozo	Clarabelle
Monsieur Sniff	Monsieur Sniff

À vos crayons ! Solution

Supposez que vous ayez des tables identiques, infos_clown1 et infos_clown2. Écrivez une jointure pour obtenir une table de résultats contenant le nom de chaque clown et celui du chef de ce clown.

infos_clown1

id	nom	id_chef
1	Elsie	3
2	Cornichon	5
3	Mme Câlin	10
4	Bobo	3
5	Clarabelle	10
6	Scooter	3
7	Zippo	3
8	Poulette	5
9	Bozo	5
10	Monsieur Sniff	10

infos_clown2

id	nom	id_chef
1	Elsie	3
2	Cornichon	5
3	Mme Câlin	10
4	Bobo	3
5	Clarabelle	10
6	Scooter	3
7	Zippo	3
8	Poulette	5
9	Bozo	5
10	Monsieur Sniff	10

À vos crayons ! Solution

Supposez que vous ayez des tables identiques, infos_clown1 et infos_clown2. Écrivez une jointure pour obtenir une table de résultats contenant le nom de chaque clown et celui du chef de ce clown.

infos_clown1

id	nom	id_chef
1	Elsie	3
2	Cornichon	5
3	Mme Câlin	10
4	Bobo	3
5	Clarabelle	10
6	Scooter	3
7	Zippo	3
8	Poulette	5
9	Bozo	5
10	Monsieur Sniff	10

infos_clown2

id	nom	id_chef
1	Elsie	3
2	Cornichon	5
3	Mme Câlin	10
4	Bobo	3
5	Clarabelle	10
6	Scooter	3
7	Zippo	3
8	Poulette	5
9	Bozo	5
10	Monsieur Sniff	10

Pour ne pas confondre les deux colonnes appelées 'nom', nous allons donner à la seconde l'alias 'chef'.

```
SELECT c1.nom, c2.nom AS chef
FROM infos_clown1 c1
INNER JOIN infos_clown2 c2
ON c1.id_chef = c2.id;
```

C'est là que nous faisons correspondre l'id_chef d'infos_clown1 avec l'id d'infos_clown2.

Nous avons besoin d'une auto-jointure

Dans l'exercice « À vos crayons » que vous venez de faire, vous aviez deux fois la même table. Mais dans une base de données normalisée, vous n'aurez jamais deux copies de la même table. À la place, on peut **utiliser une auto-jointure pour *simuler* deux tables**.

Prenez cette requête, qui est presque identique à la solution du « À vos crayons », mais comporte une différence évidente.

info_clown

id	nom	id_chef
1	Elsie	3
2	Cornichon	5
3	Mme Câlin	10
4	Bobo	3
5	Clarabelle	10
6	Scooter	3
7	Zippo	3
8	Poulette	5
9	Bozo	5
10	Monsieur Sniff	10

```
SELECT c1.nom, c2.nom AS chef
FROM info_clown c1
INNER JOIN info_clown c2
ON c1.id_chef = c2.id;
```

Nous utilisons deux fois la table info_clown sous les alias c1 (où nous prenons l'id_chef) et c2 (où nous prenons le nom du chef).

Plutôt que d'avoir deux tables identiques, nous utilisons deux fois `info_clown`, d'abord avec l'alias `c1`, puis avec l'alias `c2`. Ensuite nous réalisons une jointure interne pour relier l'`id_chef` (de `c1`) avec le `nom` du chef (de `c2`).

Cette colonne vient de la JOINTURE INTERNE d'id_chef dans la première occurence de la table info_clown (c1) et du nom de ce chef de la deuxième occurence de la table info_clown (c2).

nom	chef
Elsie	Mme Câlin
Cornichon	Clarabelle
Mme Câlin	Monsieur Sniff
Bobo	Mme Câlin
Clarabelle	Monsieur Sniff
Scooter	Mme Câlin
Zippo	Mme Câlin
Poulette	Clarabelle
Bozo	Clarabelle
Monsieur Sniff	Monsieur Sniff

L'auto-jointure vous permet d'interroger une table comme s'il y en avait deux contenant exactement les mêmes informations.

Un autre moyen d'obtenir des informations multi-tables

J'essaie d'obtenir une grande liste de toutes les professions que j'utilise dans liste_greg, mais je ne trouve pas comment faire pour lister les professions de ces trois tables en une fois.

Voici les trois tables dont parle Greg.

Emploi qu'occupe la personne pour l'instant

emploi_actuel
id_contact
profession
salaire
date_debut

Emploi que veut la personne

emploi_voulu
id_contact
profession
salaire_min
salaire_max
disponibilite
annees_exp

Emplois disponibles

offres_emplois
id_offre
profession
salaire
code_postal
description

Jusqu'ici, il a créé trois instructions SELECT distinctes :

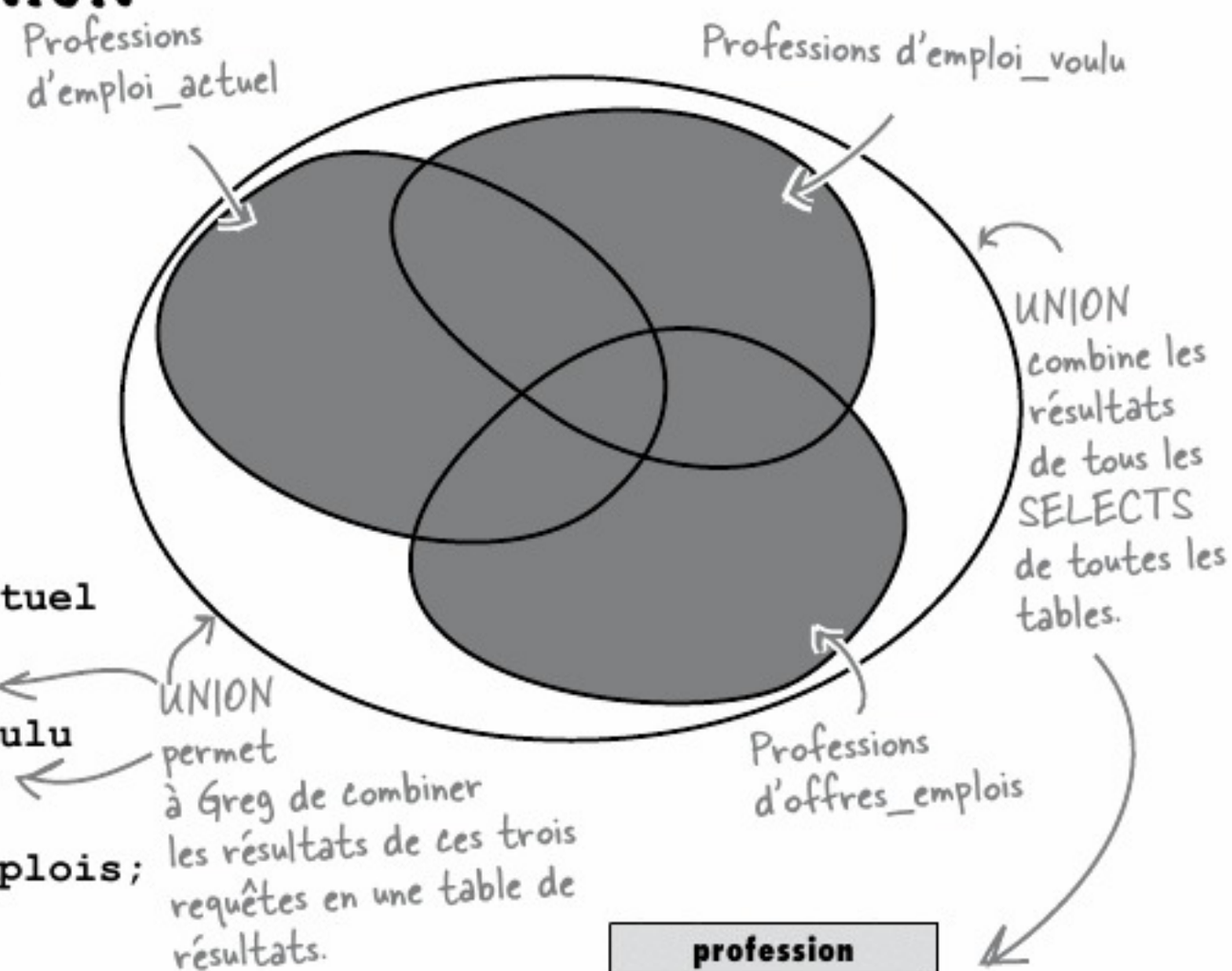
```
SELECT profession FROM emploi_actuel;
SELECT profession FROM emploi_voulu;
SELECT profession FROM offres_emplois;
```

Elles fonctionnent mais il veut combiner les résultats en une seule requête et obtenir une liste de toutes les professions de ces trois tables.

Vous pouvez utiliser une UNION

Il existe un autre moyen d'obtenir des résultats combinés de deux tables ou plus : une UNION.

Une UNION combine les résultats de deux ou plus de requêtes dans une table, à partir de ce que vous spécifiez dans la liste des colonnes du SELECT. Pensez aux résultats de l'UNION comme aux valeurs résultant de chaque SELECT, qui « se chevauchent ».




```
SELECT profession FROM emploi_actuel
UNION
SELECT profession FROM emploi_voulu
UNION
SELECT profession FROM offres_emplois;
```

UNION permet à Greg de combiner les résultats de ces trois requêtes en une table de résultats.

Voici quelques-unes des centaines de professions qu'il obtient dans les résultats combinés des trois tables.

profession
Comptable
Avocat
Programmeur
Concepteur web
Berger
Chef
Psychologue
Coiffeur
Enseignant
Serveur

Greg remarque qu'il n'y a aucun duplicat dans les résultats, mais les professions ne sont pas classées, il réessaie donc la requête avec un ORDER BY supplémentaire dans chaque instruction SELECT.

```
SELECT profession FROM emploi_actuel ORDER BY profession
UNION
SELECT profession FROM emploi_voulu ORDER BY profession
UNION
SELECT profession FROM offres_emplois ORDER BY profession;
```

Greg a ajouté un ORDER BY à chaque instruction pour que les professions de la table des résultats soient classées dans l'ordre alphabétique.

MUSCLEZ VOS NEURONES

Que se passe-t-il quand Greg exécute cette nouvelle requête ?

UNION est limitée

La requête de Greg n'a pas fonctionné ! Greg a obtenu une erreur, parce que son logiciel ne savait pas comment interpréter l'`ORDER BY` plusieurs fois.

`UNION` ne peut prendre **un `ORDER BY` qu'à la *fin* de l'instruction**. Ceci parce qu'`UNION` **concatène et regroupe les résultats de plusieurs instructions `SELECT`.**

Il y a quelques autres choses à savoir sur les unions.

Règles SQL de l'UNION

Le **nombre de colonnes** de chaque instruction `SELECT` **doit correspondre**. Vous ne pouvez pas sélectionner deux colonnes dans la première instruction et une dans la suivante.

Vous devez aussi avoir les mêmes expressions et fonctions aggrégées dans chaque instruction `SELECT`.

Vous pouvez placer les instructions `SELECT` dans n'importe quel ordre ; cela ne changera pas les résultats.

Règles SQL de l'UNION

Par défaut, SQL supprime les valeurs dupliquées des résultats d'une union.

Les types de données des colonnes doivent être soit les mêmes, soit convertibles de l'un à l'autre.

Si, pour une raison ou une autre, vous **VOULEZ** voir les duplicats, vous pouvez utiliser l'opérateur `UNION ALL`. Il renvoie toutes les correspondances, pas seulement les correspondances distinctes.

Les règles de l'UNION en action

Le nombre de colonnes dans les instructions SELECT que vous combinez avec UNION doivent correspondre. Vous ne pouvez pas SELECTionner deux colonnes de la première table et une seule colonne de la table suivante.

Vous devez utiliser le même nombre de colonnes dans chaque SELECT.

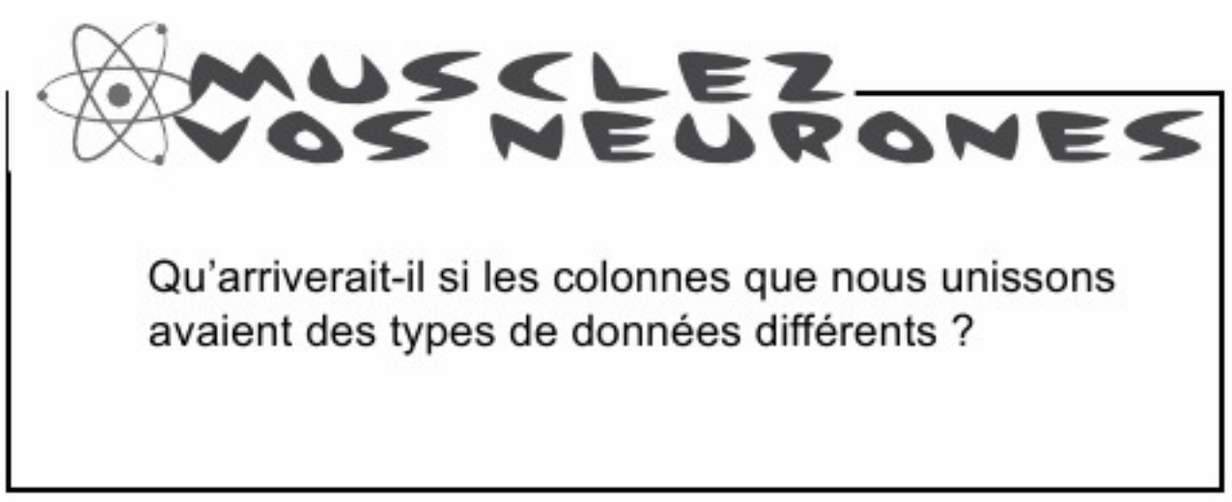
```
SELECT profession FROM emploi_actuel
UNION
SELECT profession FROM emploi_voulu
UNION
SELECT profession FROM offres_emplois
ORDER BY profession;
```

Si vous voulez classer vos résultats, utilisez un ORDER BY après le dernier SELECT que vous combinez. Cela classe la liste de résultats toute entière.

profession
Avocat
Berger
Bijoutier
Boulanger
Clown
Coiffeur
Mécanicien
Médecin
Neurochirurgien
Orthopédiste

Voici un exemple des résultats que l'on peut s'attendre à recevoir.

Dans cet exemple, les trois colonnes ont le même type de données, VARCHAR. En conséquence, la colonne renvoyée par la requête est aussi une VARCHAR.



Qu'arriverait-il si les colonnes que nous unissons avaient des types de données différents ?

UNION ALL

UNION ALL fonctionne exactement de la même façon qu'UNION, sauf qu'il renvoie toutes les valeurs des colonnes au lieu de renvoyer une occurence de chaque valeur dupliquée.

```
SELECT profession FROM emploi_actuel
UNION ALL
SELECT profession FROM emploi_voulu
UNION ALL
SELECT profession FROM offres_emplois
ORDER BY profession;
```

Cette fois nous voulons voir toutes les valeurs conservées dans les colonnes profession des trois tables.

profession
Avocat
Avocat
Berger
Bijoutier
Boulanger
Boulanger
Boulanger
Clown
Clown
Clown
Coiffeur
Dresseur
Mécanicien
Mécanicien
Médecin
Neurochirurgien
Orthopédiste

Cette fois nous obtenons la même profession plus d'une fois.

Jusqu'ici nos UNION ont utilisé des colonnes du même type de données. Mais vous pouvez avoir besoin de créer une UNION de colonnes avec des types de données différents.

Quand nous disons que les types de données doivent être convertibles de l'un à l'autre, cela signifie que les types de données renvoyés vont être convertis en types compatibles si c'est possible, et si ce n'est pas possible, la requête va échouer.

Supposez que vous utilisez une UNION sur un type de données INTEGER et un type de données VARCHAR. Comme VARCHAR ne peut pas devenir un entier, les lignes de résultat vont convertir l'INTEGER en VARCHAR.

Créer une table à partir d'une union

Nous ne pouvons pas voir facilement le type de données renvoyé par notre UNION, à moins de le capturer. Nous pouvons utiliser un CREATE TABLE AS pour récupérer les résultats de notre UNION et les regarder de plus près.

L'instruction CREATE TABLE AS prend les résultats d'une requête SELECT et en fait une table. Dans l'exemple ci-dessous, nous plaçons les professions de notre UNION dans une nouvelle table appelée mon_union.

Le nom de notre nouvelle table

```
CREATE TABLE mon_union AS
SELECT profession FROM emploi_actuel UNION
SELECT profession FROM emploi_voulu
UNION SELECT profession FROM offres_emplois;
```

C'est l'UNION que vous avez déjà vue. Vous pouvez créer une table à partir de n'importe quelle instruction SELECT.

À vos crayons !

Créez une UNION avec : id_contact d'emploi_actuel et salaire d'offres_emplois

. .....

. .....

Devinez quel va être le type de données des résultats, puis écrivez une instruction CREATE TABLE AS avec votre UNION.

. .....

. .....

. .....

Utilisez DESC sur votre table pour voir si vous aviez raison pour le type de données.

. .....

Réponses page 453.

INTERSECT et EXCEPT

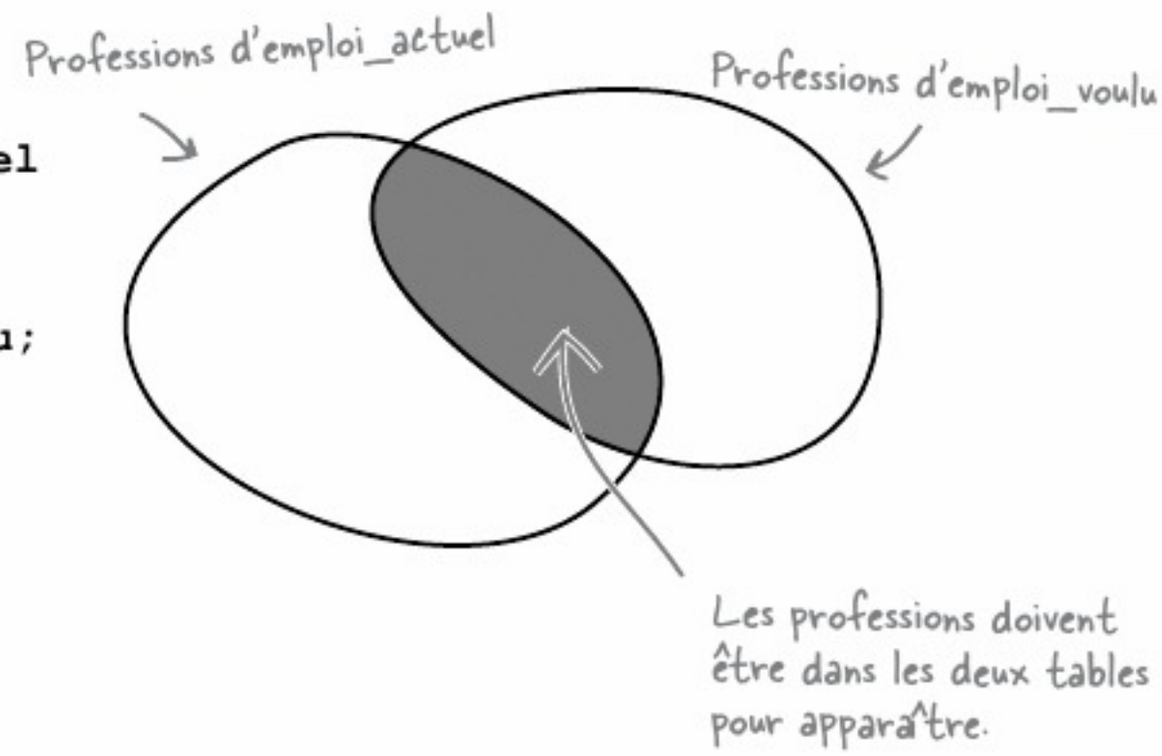
INTERSECT et EXCEPT sont utilisés d'une façon similaire à UNION : pour trouver des parties de requêtes qui se chevauchent.

INTERSECT ne renvoie que les colonnes qui se trouvent dans la première et dans la deuxième requête.

Ouvrez les yeux !

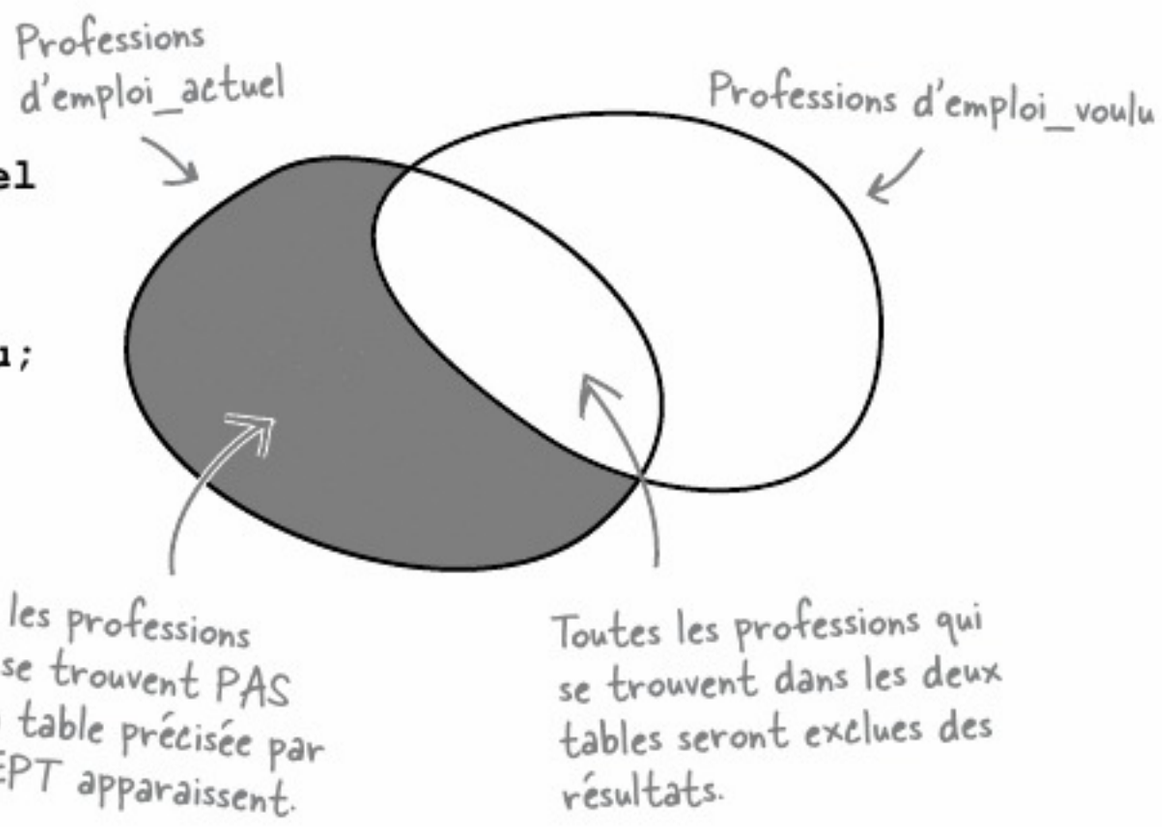
Ces deux opérations N'EXISTENT PAS dans MySQL.

```
SELECT profession FROM emploi_actuel
INTERSECT
SELECT profession FROM emploi_voulu;
```

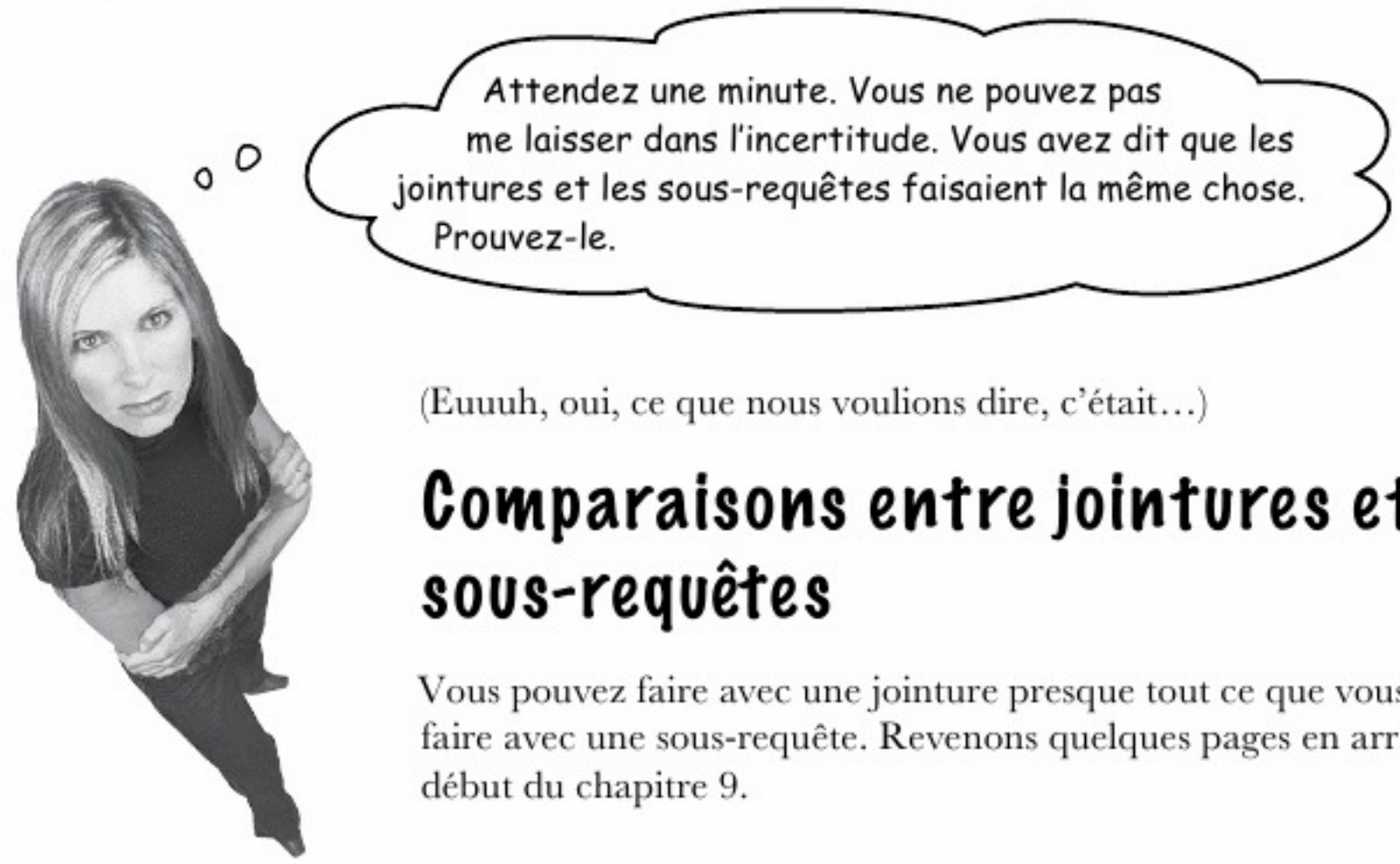


EXCEPT ne renvoie que les colonnes qui se trouvent dans la première requête mais **pas** dans la seconde.

```
SELECT profession FROM emploi_actuel
EXCEPT
SELECT profession FROM emploi_voulu;
```



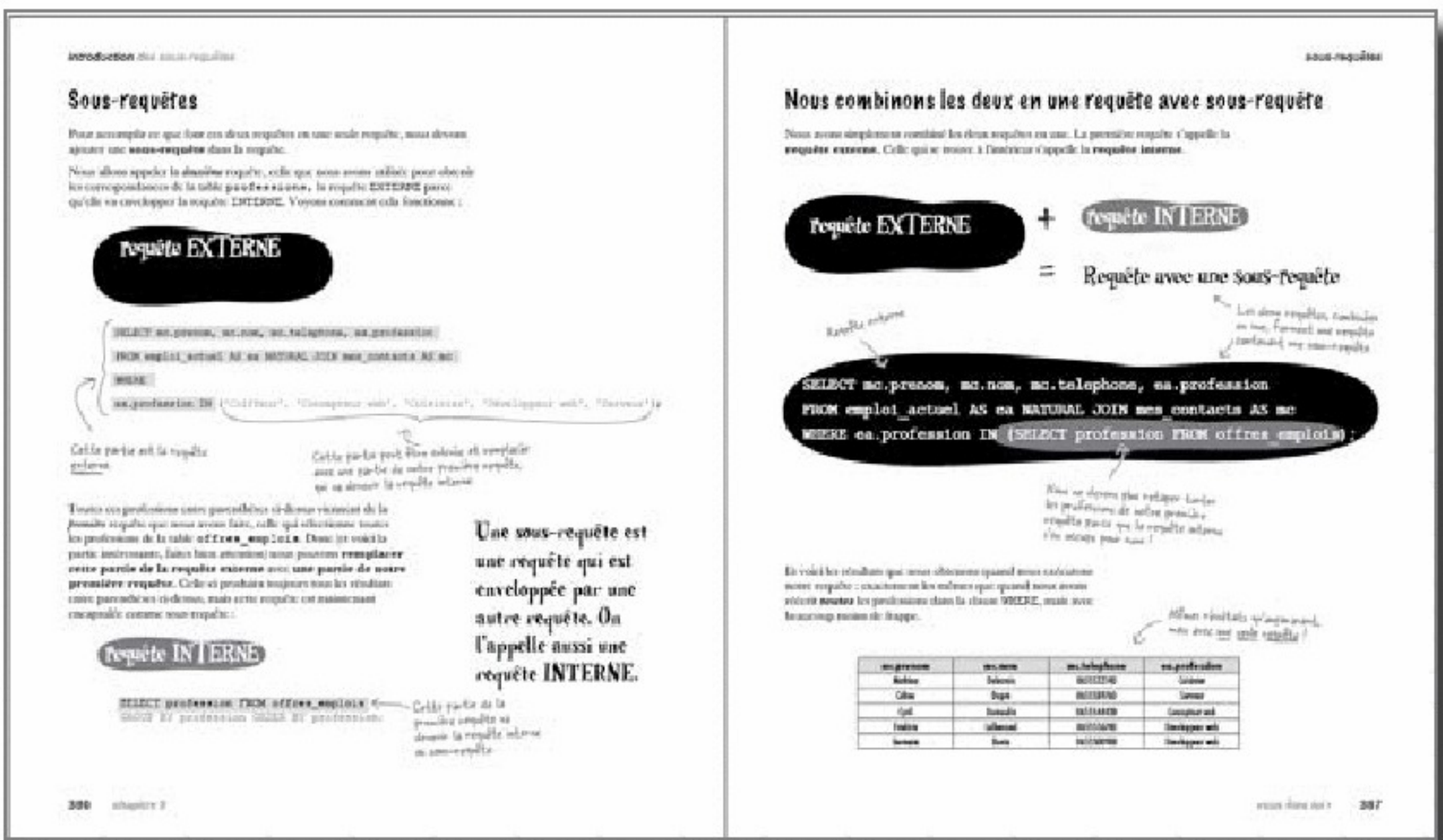
Nous en avons fini avec les jointures, il est temps de passer à...



(Euuuh, oui, ce que nous voulions dire, c'était…)

Comparaisons entre jointures et sous-requêtes

Vous pouvez faire avec une jointure presque tout ce que vous pouvez faire avec une sous-requête. Revenons quelques pages en arrière, au début du chapitre 9.



Sous-requêtes

Requête EXTERNE

Requête INTERNE

Une sous-requête est une requête qui est enveloppée par une autre requête. On l'appelle aussi une requête INTERNE.

Nous combinons les deux en une requête avec sous-requête

Requête EXTERNE + Requête INTERNE = Requête avec une sous-requête

```
SELECT mc.prenom, mc.nom, mc.telephone, ea.profession
FROM emploi_actuel AS ea NATURAL JOIN mes_contacts AS mc
WHERE ea.profession IN (SELECT profession FROM offres_emploi);
```

Changer une sous-requête en jointure

Dans le chapitre 9, voici la première sous-requête que nous avons créée :

Requête externe

```
SELECT mc.prenom, mc.nom, mc.telephone, ea.profession
FROM emploi_actuel AS ea NATURAL JOIN mes_contacts AS mc
WHERE ea.profession IN (SELECT profession FROM offres_emplois);
```

Requête interne

Et voici les résultats que nous obtenons quand nous exécutons notre requête :

mc.prenom	mc.nom	mc.telephone	ea.profession
Mathieu	Delecroix	0655532140	Cuisinier
Céline	Dupré	0655589760	Serveur
Cyril	Dumoulin	0655544430	Concepteur web
Frédéric	Lallemand	0655556740	Développeur web
Germain	Bonte	0655500980	Développeur web

À vos crayons !

Voici la clause WHERE avec la sous-requête, réécrite avec un INNER JOIN:

```
SELECT mc.prenom, mc.nom, mc.telephone, ea.profession
FROM emploi_actuel AS ea NATURAL JOIN mes_contacts AS mc
INNER JOIN offres_emplois oe
ON ea.profession = oe.profession;
```

Vous pouvez remplacer le WHERE contenant la sous-requête par un INNER JOIN.

Expliquez pourquoi cette partie INNER JOIN de la requête vous obtient les mêmes résultats que la sous-requête.

.....

.....

.....

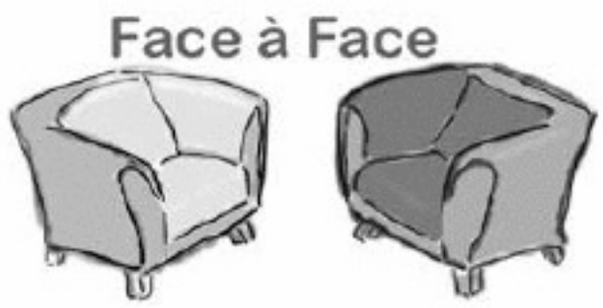
Laquelle de ces requêtes est la plus facile à comprendre, selon vous ?

.....

Réponses page 453.

Non, si ces sous-requêtes font ce que vous voulez qu'elles fassent, vous n'avez pas besoin de les réécrire.

Mais il existe certainement des raisons de choisir parfois l'une plutôt que l'autre...



Le débat de ce soir : **Jointure ou Sous-requête, laquelle est la meilleure**

Jointure

Je suis évidemment le meilleur choix dans la plupart des cas. Je suis plus facile à comprendre et généralement, je m'exécute beaucoup plus vite que la vieille Sous-requête là-bas.

Je m'en sortais très bien sans vous. Je suis plus facile à comprendre que vous.

C'est vous qui le dites. Et vos incongruïtés de CORRÉLÉE et NON-CORRÉLÉE ?

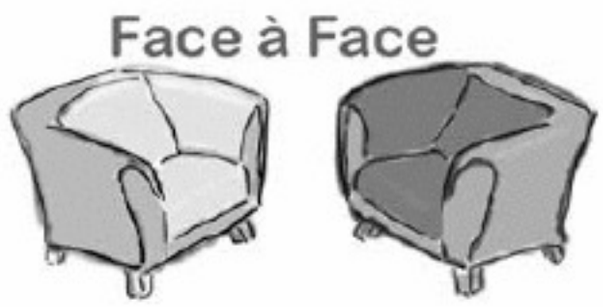
Sous-requête

Excusez-moi ? Qui appelez-vous « vieille » ? Je ne suis même apparue qu'assez tard dans certains SGBDR. J'ai été AJOUTÉE à cause du nombre de programmeurs qui voulaient m'utiliser.

Qui essayez-vous de berner, avec vos blablabla d'INNER et d'OUTER ? Tout ça est déconcertant…

D'accord, nous avons toutes les deux notre propre jargon, c'est vrai. Mais avec moi, vous pouvez normalement comprendre la partie interne puis la partie externe séparément.

→ À suivre page suivante.



Le débat de ce soir : **Jointure ou Sous-requête, laquelle est meilleure**

Jointure

Pas toujours, Mme la Sous-requête CORRÉLÉE. Mais d'accord, laissons cela de côté pour l'instant. Je suis la meilleure option quand on a besoin de colonnes de plusieurs tables dans les résultats. En fait, je suis la seule option dans ce cas-là.

C'est possible, mais il n'est pas si difficile de comprendre ce que je fais. Vous, vous ne pouvez même pas utiliser d'alias pour éviter de retaper les noms des tables encore et encore.

Lalala. Madame est trop bien pour avoir des alias, alors ? Vous pensez que vous êtes tellement plus simple que moi, mais, et ces sous-requêtes corrélées ? Elles sont aussi tortueuses que ce que je fais.

Frimeuse.

Sous-requête

C'est bien pour ça que vous n'êtes pas si douée avec les valeurs agrégées. Vous ne pouvez pas utiliser des agrégats dans une clause WHERE sans une sous-requête. Cela compense un peu pour ne pas renvoyer plusieurs colonnes. Vous êtes si compliquée.

Oui, en fait, ces alias, je pense qu'ils rendent les choses encore plus difficiles à suivre. Et, sachez-le, je peux les utiliser aussi. Mais quand je les utilise, c'est beaucoup plus direct. La plupart du temps je ne m'encombre même pas d'alias.

Euuh... c'est vrai. Mais il y a une chose qui me rend très différente de vous. Je peux être utilisée avec UPDATE, INSERT et DELETE.

Prenez ces requêtes et les sous-requêtes du chapitre 9 et voyez si vous pouvez les écrire sans sous-requêtes ou s'il vaut tout simplement mieux laisser les sous-requêtes dans vos requêtes. Les jointures sont autorisées.

Dressez la liste des professions dont le salaire est égal au plus haut salaire de la table offres_emplois.

```
SELECT profession FROM offres_emplois WHERE salaire =
(SELECT MAX(salaire) FROM offres_emplois);
```

. .....

. .....

. .....

Vaut-il mieux utiliser des sous-requêtes ? . .....

Dressez la liste des noms et prénoms des gens avec un salaire supérieur au salaire moyen.

```
SELECT mc.prenom, mc.nom FROM mes_contacts mc NATURAL
JOIN emploi_actuel ea WHERE ea.salaire > (SELECT
AVG(salaire) FROM emploi_actuel);
```

. .....

. .....

. .....

Vaut-il mieux utiliser des sous-requêtes ? . .....

Prenez ces requêtes et les sous-requêtes du chapitre 9 et voyez si vous pouvez les écrire sans sous-requêtes ou s'il vaut tout simplement mieux laisser les sous-requêtes dans vos requêtes. Les jointures sont autorisées.

Dressez la liste des professions dont le salaire est égal au plus haut salaire de la table offres_emplois.

```
SELECT profession FROM offres_emplois WHERE salaire =
(SELECT MAX(salaire) FROM offres_emplois);
```

SELECT profession FROM offres_emplois ORDER BY salaire DESC LIMIT 1;

Ceci permet à la requête de ne renvoyer qu'un seul résultat, la ligne avec le plus grand salaire.

Vaut-il mieux utiliser des sous-requêtes ?. Non.

Dressez la liste des noms et prénoms des gens avec un salaire supérieur au salaire moyen.

```
SELECT mc.prenom, mc.nom FROM mes_contacts mc NATURAL
JOIN emploi_actuel ea WHERE ea.salaire > (SELECT
AVG(salaire) FROM emploi_actuel);
```

Oh oh, nous ne pouvons pas utiliser LIMIT et ORDER BY pour obtenir des moyennes comme nous l'avons fait ci-dessus.

Vaut-il mieux utiliser des sous-requêtes ?. Oui.

Dans la solution précédente, nous avons pu utiliser LIMIT pour obtenir le plus grand salaire d'une liste triée de salaires. Nos salaires supérieurs-à-la-moyenne ne peuvent pas être triés, nous ne pouvons donc pas utiliser LIMIT pour les récupérer.

Une auto-jointure comme sous-requête

Puisque vous avez vu comment transformer une sous-requête en jointure, regardons comment transformer une auto-jointure en sous-requête.

Vous souvenez-vous de l'`id_chef` de clown que nous avons ajouté à notre table `info_clown` ? Voici l'auto-jointure que nous avons utilisée, où nous avons appelé une occurence de la table `info_clown` `c1` et la seconde `c2`.

Indique quel clown est le chef de quel clown

info_clown

id	nom	id_chef
1	Elsie	3
2	Cornichon	5
3	Mme Câlin	10
4	Bobo	3
5	Clarabelle	10
6	Scooter	3
7	Zippo	3
8	Poulette	5
9	Bozo	5
10	Monsieur Sniff	10

AVANT

```
SELECT c1.nom, c2.nom AS chef
FROM info_clown c1
INNER JOIN info_clown c2
ON c1.id_chef = c2.id;
```

La première occurence d'info_clown

La deuxième occurence d'info_clown

APRÈS

Quand nous transformons une auto-jointure en sous-requête, la sous-requête est CORRÉLÉE puisqu'elle dépend du résultat de la requête externe pour obtenir le bon `id_chef` et elle apparaît dans la liste des colonnes du `SELECT`.

Requête externe.

La sous-requête est dans la liste du SELECT.

```
SELECT c1.nom,
(SELECT nom FROM info_clown
WHERE c1.id_chef = id) AS chef
FROM info_clown c1;
```

La sous-requête dépend du résultat de la requête externe pour obtenir le bon id_chef, elle est donc corrélée.

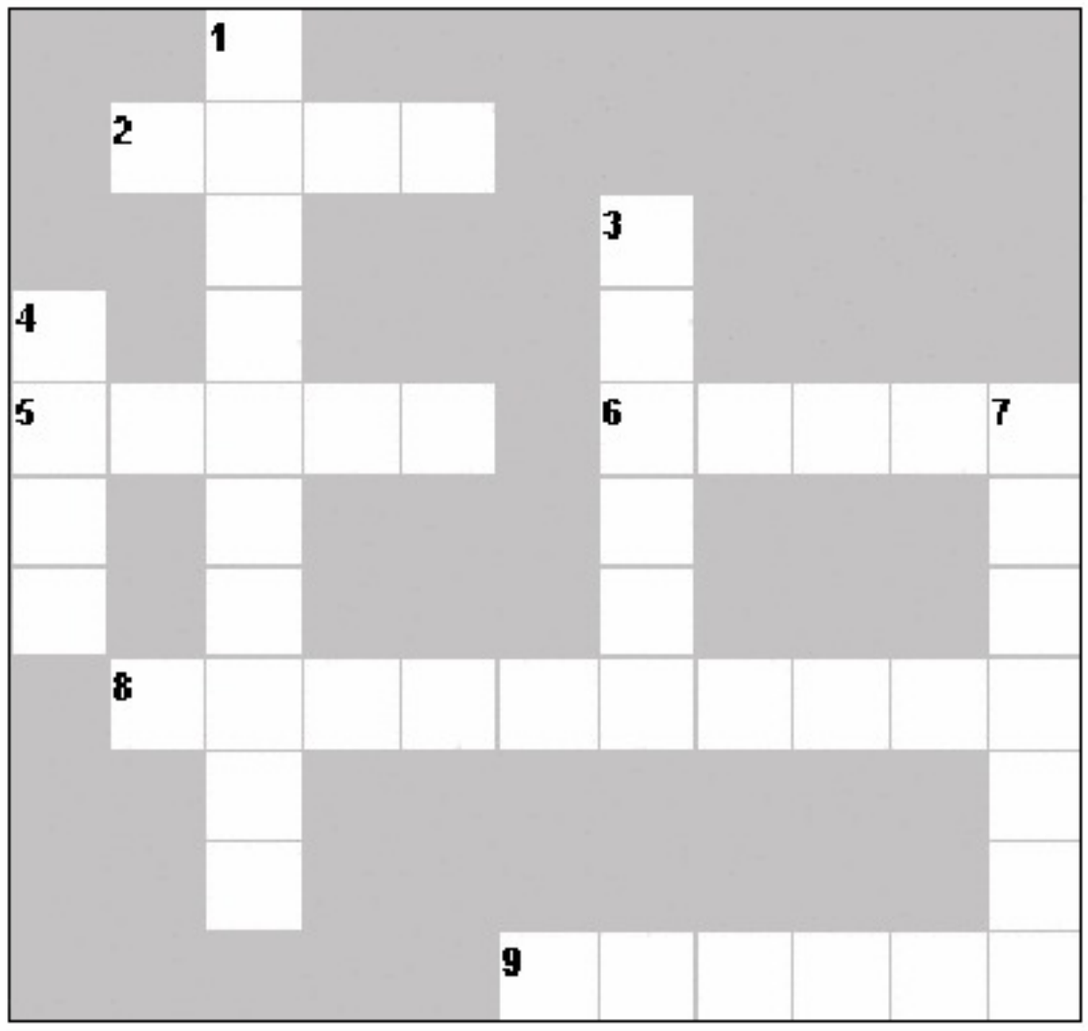
L'entreprise de Greg se développe

Greg est très occupé à découvrir les jointures et les sous-requêtes. Il a engagé des amis pour l'aider avec les requêtes moins complexes.

Malheureusement, ils ne savent pas ce qu'ils font. Greg est sur le point de découvrir ce qui arrive quand plusieurs personnes aux compétences SQL incertaines travaillent sur la même base de données en même temps.

Jointures-et-Unions-croisées

Ce fut un chapitre turbo-chargé, avec beaucoup à apprendre. Aidez-vous à tout mémoriser en faisant ces mots-croisés. Toutes les réponses viennent du chapitre.



Définitions horizontales :

2. On peut utiliser une ...-jointure pour simuler que l'on a deux tables.

5. Cela combine les résultats de deux requêtes ou plus en une table, à partir de ce que vous précisez dans la liste des colonnes du SELECT.

6. Avec une jointure interne, vous comparez les lignes de deux tables, mais l'... de ces tables n'a pas d'importance.

8. Une clé étrangère auto-... est la clé primaire de la table, utilisée dans cette même table dans un autre but.

9. Une jointure externe ... prend toutes les lignes de la table de gauche et les fait correspondre à celle de la table de DROITE.

Définitions verticales :

1. Par défaut, SQL supprime les valeurs ... des résultats d'une union.

3. La jointure externe ... compare la table de droite à la table de gauche.

4. Dans une jointure externe gauche, cela signifie que la table de droite n'a pas de valeur correspondant à la table de gauche.

7. Une jointure ... vous donne une ligne, qu'il y ait une correspondance avec l'autre table ou pas.

Boîte à outils SQL

SQL est maintenant une promenade pour vous. Vous avez vu les jointures externes, les auto-jointures et les unions. Vous savez même comment convertir une jointure en sous-requête et inversement. Pour une liste exhaustive des outils du livre, voyez l'annexe C.

CLÉ ÉTRANGÈRE AUTO-RÉFÉRENCÉE

C'est une clé étrangère dans la même table où elle est clé primaire, mais utilisée dans un autre but.

JOINTURE EXTERNE GAUCHE

Un LEFT OUTER JOIN prend toutes les lignes de la table de gauche et les fait correspondre à celles de la table de DROITE.

JOINTURE EXTERNE DROITE

Un RIGHT OUTER JOIN prend toutes les lignes de la table de droite et les fait correspondre à celles de la table de GAUCHE

AUTO-JOINTURE

L'AUTO-JOINTURE vous permet d'interroger une table comme s'il y en avait deux, contenant exactement les mêmes informations.

UNION et UNION ALL

Une UNION combine les résultats de deux ou plus de requêtes dans une table, à partir de ce que vous précisez dans la liste des colonnes du SELECT.

UNION cache les valeurs dupliquées, UNION ALL les inclut.

INTERSECT

Utilisez ce mot-clé pour ne renvoyer que des valeurs se trouvant dans la première ET la seconde requête

EXCEPT

Utilisez ce mot-clé pour ne renvoyer que les valeurs de la première requête ne se trouvant PAS dans la seconde.

CREATE TABLE AS

Utilisez cette commande pour créer une table à partir des résultats d'une instruction SELECT.

À vos crayons ! Solution

De la page 441.

Créez une UNION avec : id_contact d'emploi_actuel et salaire d'offres_emplois

SELECT id_contact FROM emploi_actuel UNION
SELECT salaire FROM offres_emplois;

Devinez quel va être le type de données des résultats, puis écrivez une instruction CREATE TABLE AS avec votre UNION.

CREATE TABLE ma_table AS SELECT id_contact FROM emploi_actuel UNION SELECT salaire FROM offres_emplois;

Utilisez DESC sur votre table pour voir si vous aviez raison pour le type de données.

DEC(12,2)

À vos crayons ! Solution

De la page 444.

Voici la clause WHERE avec la sous-requête, réécrite avec un INNER JOIN:

```
SELECT mc.prenom, mc.nom, mc.telephone, ea.profession
FROM emploi_actuel AS ea NATURAL JOIN mes_contacts AS mc
INNER JOIN offres_emplois oe
ON ea.profession = oe.profession;
```

Vous pouvez remplacer le WHERE contenant la sous-requête par un INNER JOIN.

Expliquez pourquoi cette partie INNER JOIN de la requête vous obtient les mêmes résultats que la sous-requête.

L'INNER JOIN montre seulement les résultats où ea.profession = oe.profession, ce qui est équivalent à la clause WHERE avec sous-requête :
WHERE ea.profession IN (SELECT profession FROM offres_emplois);

Laquelle de ces requêtes est la plus facile à comprendre, selon vous ? Il n'y a pas de bonne réponse ici ! Mais votre réponse montre que vous commencez à réfléchir à ce que vous pourrez utiliser dans le futur avec vos propres données.

Jointures-et-Unions-croisées
Solution

		1 D								
	2 A	U	T	O						
		P				3 D				
4 N		L				R				
5 U	N	I	O	N		6 O	R	D	R	7 E
L		Q				I				X
L		U				T				T
	8 R	É	F	É	R	E	N	C	É	E
		E								R
		S								N
					9 G	A	U	C	H	E

11 contraintes, vues et transactions

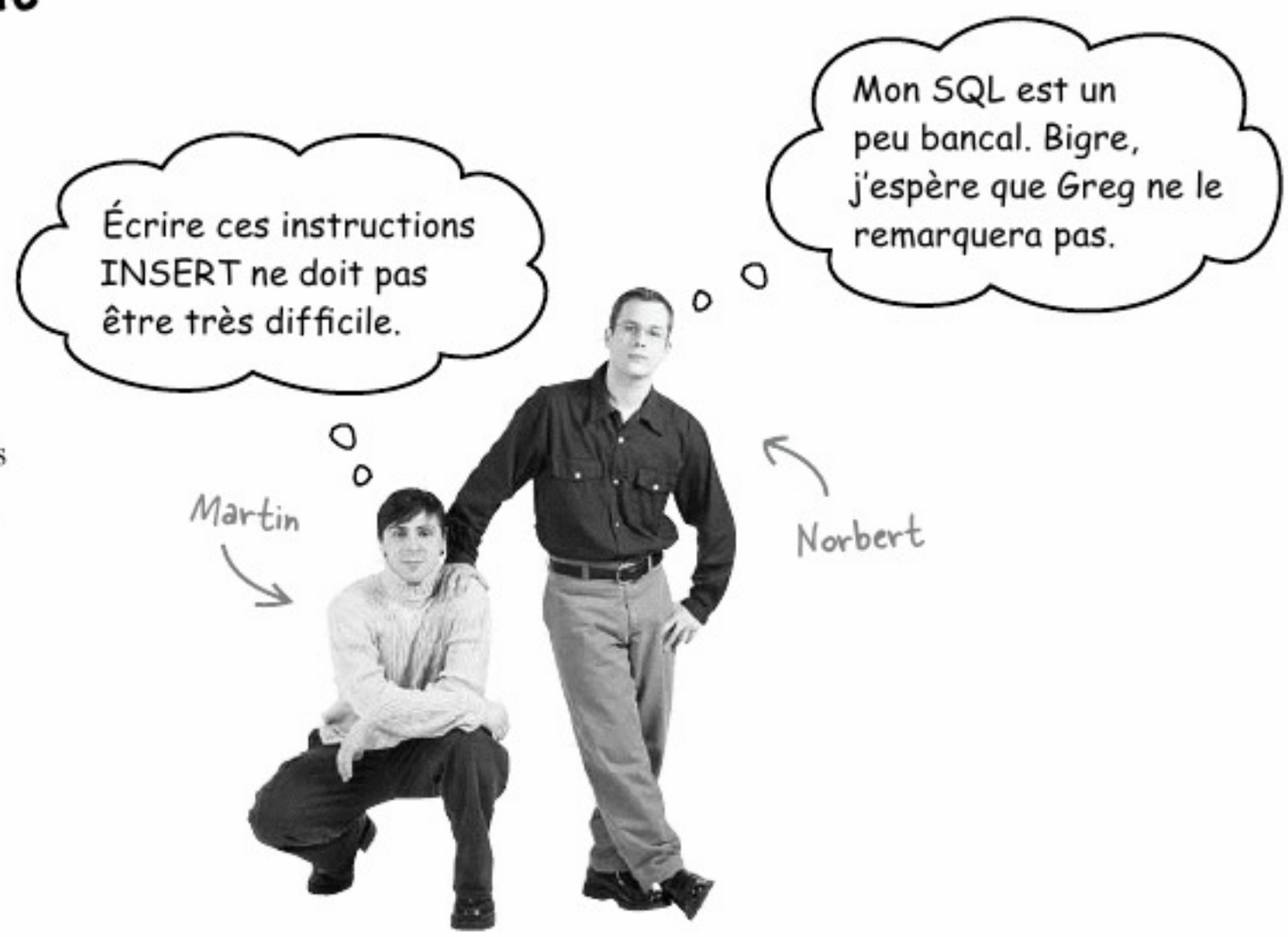
Trop de cuisiniers gâtent la base de données

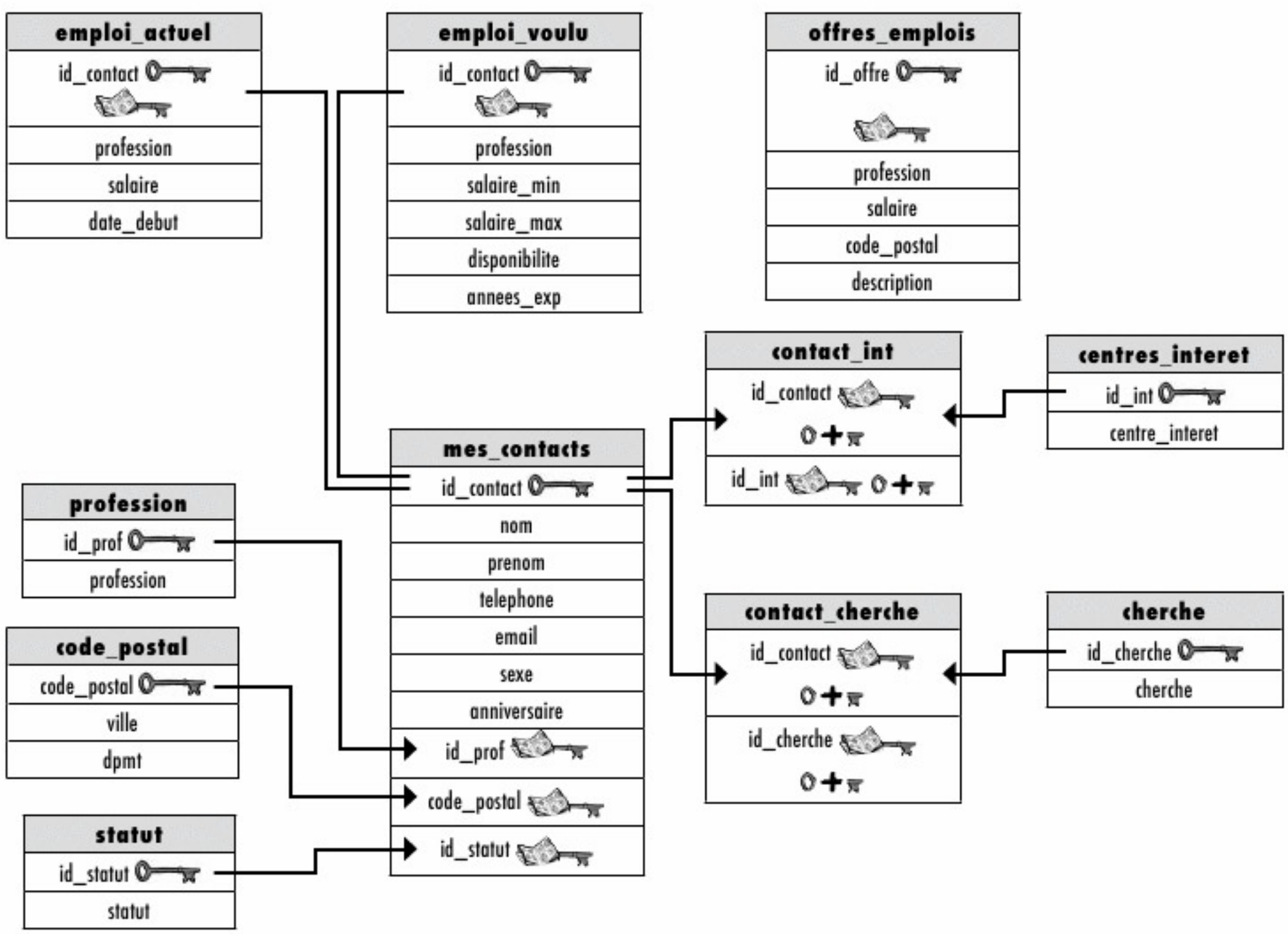
Votre base de données a grandi et d'autres que vous doivent l'utiliser. Le problème est que certains d'entre eux ne sont pas aussi compétents que vous en SQL. Vous avez besoin de moyens de **les empêcher de saisir de mauvaises données**, de techniques pour ne les autoriser à **voir qu'une partie des données** et de méthodes pour **les empêcher de se marcher les uns sur les autres quand ils essaient d'entrer des données en même temps**. Dans ce chapitre, nous commençons à protéger vos données des erreurs des autres. Bienvenue dans les Bases de données défensives, 1ère partie.

Greg a recruté de l'aide

Greg a engagé deux personnes pour l'aider à gérer son entreprise en expansion. Martin va s'occuper de saisir les nouveaux clients dans la base de données, pendant que Norbert sera chargé de faire correspondre les gens à des emplois potentiels.

Greg leur a expliqué sa base de données et leur a décrit ce que fait chaque table.





Le premier jour de Martin : insertion d'un nouveau client

Martin est assis à son nouveau bureau et reçoit un message instantané de Greg :

Chat de l'équipe : Voici les données à insérer

Greg : Salut Martin, pouvez-vous ajouter quelqu'un à la bdd pour moi ?

Martin : Oui, bien sûr.

Greg : Il n'y a qu'une partie des infos, je vous envoie le reste plus tard :

Greg : Camille Dubuc, 0600331239, camilledubuc@unemail.com, code postal 45000

Greg : La date de naissance est le 15/04/1978.

Greg : Pour profession, mettez enseignant et le statut est célibataire (vous devrez faire quelques SELECT pour obtenir les bonnes valeurs, regardez mes notes pour la syntaxe)

Martin : Ça semble assez simple. Je m'y mets. :)

Greg : Merci !

De rien |

MUSCLEZ VOS NEURONES

Pouvez-vous écrire les requêtes pour insérer cette nouvelle personne dans la base de données ?

Martin évite un NULL

On m'a dit qu'il valait mieux éviter les NULL, mais je ne connais pas le sexe pour cette entrée.

En entrant les données, Martin réalise qu'il ne sait pas si Camille est un homme ou une femme. Greg n'est pas là, il décide donc de faire preuve d'initiative et de saisir 'X' pour sexe.

Voici ses requêtes :

mes_contacts
id_contact
nom
prenom
telephone
email
sexe
anniversaire
id_prof
code_postal
id_statut

Il récupère l'id_prof de la table profession

```
SELECT id_prof FROM profession WHERE profession = 'Enseignant';
```

id_prof
17

Voici l'id qui correspond à 'Enseignant', il peut donc l'utiliser dans sa requête pour mes_contacts.

Il récupère l'id_statut de la table statut

```
SELECT id_statut FROM statut WHERE statut = 'célibataire';
```

id_statut
1

Et voici l'id_statut qui correspond à 'célibataire'.

Il insère ces valeurs et utilise X pour sexe

Quand nous avons une colonne AUTO_INCRÉMENTÉE (id_contact), nous n'avons pas besoin d'y mettre une valeur.

```
INSERT INTO mes_contacts (nom, prenom, telephone, email, sexe, anniversaire,
id_prof, code_postal, id_statut) VALUES('Dubuc', 'Camille', '0600331239',
'camilledubuc@unemail.com', 'X', '1978-04-15', 17, '45000', 1);
```

Voilà ce que Martin décide de saisir pour sexe, plutôt que d'essayer de deviner ou d'entrer NULL.

Voici les id qu'il a trouvé avec les deux requêtes ci-dessus. Il aurait pu faire cela avec des sous-requêtes.

Bond de trois mois dans le futur

Greg essaie d'établir des données démographiques. Il veut savoir combien de personnes dans `mes_contacts` sont des hommes, combien sont des femmes et combien d'entrées il a au total. Il écrit trois requêtes : d'abord il récupère un nombre de femmes et d'hommes, puis il récupère un nombre total.

```
SELECT COUNT(*) AS Femmes FROM mes_contacts WHERE sexe = 'F';
```

Femmes
5975

Greg découvre qu'il a 5 975 lignes avec 'F' pour sexe dans sa table mes_contacts.

```
SELECT COUNT(*) AS Hommes FROM mes_contacts WHERE sexe = 'H';
```

Hommes
6982

Et 6 982 valeurs 'H' pour sexe.

```
SELECT COUNT(*) AS Total FROM mes_contacts;
```

Total
12970

Il vérifie le nombre total de lignes dans sa table avec cette requête.

Greg remarques que les nombres ne correspondent pas. Il a 13 lignes qui n'apparaissent ni dans la requête des hommes ni dans celle des femmes. Il essaie une autre requête :

```
SELECT sexe FROM mes_contacts
WHERE sexe <> 'H' AND sexe <> 'F';
```

sexe
X
X
X
X
X
X
X
X
X
X
X
X
X

En cherchant les enregistrements manquants, il repère les valeurs 'X' dans sexe.

MUSCLEZ VOS NEURONES

Comment Martin aurait-il pu éviter les valeurs X ?

Par CHECK, s'il vous plaît : ajouter une CONTRAINTE CHECK

Nous avons déjà vu quelques contraintes sur les colonnes dans de précédents chapitres. Une **contrainte** est une restriction sur ce que vous pouvez insérer dans une colonne. Les contraintes sont ajoutées lorsque nous créons une table. Les contraintes que vous avez déjà vues comprennent `NOT NULL`, `PRIMARY KEY`, `FOREIGN KEY` et `UNIQUE`.

Il existe une autre sorte de contrainte de colonne, appelée **`CHECK`** (vérifie). En voici un exemple. Supposez que vous ayez une tirelire et que nous voulions faire un suivi des pièces qui y sont insérées. Elle ne prend que les pièces de dix, vingt et cinquante centimes et celles d'un et de deux euros. Nous pouvons utiliser les nombres correspondants pour représenter chaque type de pièce. La table ci-dessous utilise une contrainte `CHECK` pour restreindre les valeurs qui peuvent être insérées dans la colonne pièce :

Une contrainte CHECK restreint les valeurs que vous pouvez insérer dans une colonne. Elle utilise les mêmes conditionnels qu'une clause WHERE.

```
CREATE TABLE tirelire
(
  id INT AUTO_INCREMENT NOT NULL PRIMARY KEY,
  piece DEC(3,2) CHECK (piece IN (2,1,0.5,0.2,0.1))
);
```

Ceci vérifie si la valeur de la colonne piece est l'une de celles-ci.

Si la valeur que vous essayez d'insérer ne remplit pas la condition CHECK, vous obtenez une erreur.

Ouvrez les yeux !

CHECK ne fait pas respecter l'intégrité des données dans MySQL.

Vous pouvez créer vos tables avec des contraintes CHECK sous MySQL, mais cela ne vous aidera pas. MySQL les ignore.

CHECK pour vérifier le sexe

Si Greg pouvait retourner dans le passé, il aurait créé `mes_contacts` avec une contrainte `CHECK` sur la colonne sexe. Aujourd'hui, il peut réparer cela avec un `ALTER TABLE`.

```
ALTER TABLE mes_contacts
ADD CONSTRAINT CHECK sexe IN ('H','F');
```

Le jour suivant, Martin se retrouve dans l'incapacité de saisir 'X' pour sexe. Quand il interroge Greg sur le sujet, ce dernier lui explique la nouvelle contrainte et lui dit que puisqu'il ne peut pas revenir en arrière, Martin va devoir contacter toutes les personnes de sexe 'X' pour trouver ce qu'ils devraient être.

À vos crayons !

Notez les valeurs qui sont autorisées dans chacune de ces colonnes selon vous.

```
CREATE TABLE table_mystere
(
  colonne1 INT(4) CHECK (colonne1 > 200),

  colonne2 CHAR(1) CHECK (colonne2 NOT IN ('x', 'y', 'z')),

  colonne3 VARCHAR(3) CHECK ('A' = SUBSTRING(colonne3, 1, 1)),

  colonne4 VARCHAR(3) CHECK ('A' = SUBSTRING(colonne4, 1, 1)
  AND '9' = SUBSTRING(colonne4, 2, 1))
)
```

Colonne 1 : . .....

Colonne 2 : . .....

Colonne 3 : . .....

Colonne 4 : . .....

À vos crayons ! Solution

Notez les valeurs qui sont autorisées dans chacune de ces colonnes selon vous.

```
CREATE TABLE table_mystere
(
  colonne1 INT(4) CHECK (colonne1 > 200),

  colonne2 CHAR(1) CHECK (colonne2 NOT IN ('x', 'y', 'z')),

  colonne3 VARCHAR(3) CHECK ('A' = SUBSTRING(colonne3, 1, 1)),

  colonne4 VARCHAR(3) CHECK ('A' = SUBSTRING(colonne4, 1, 1)
  AND '9' = SUBSTRING(colonne4, 2, 1))
)
```

Vous pouvez combiner des conditions avec AND et OR.

Colonne 1 : Les valeurs insérées doivent être supérieures à 200

Colonne 2 : Tous les caractères autres que x, y et z peuvent être insérés

Colonne 3 : Le premier caractère de la chaîne doit être A

Colonne 4 : Le premier caractère de la chaîne doit être A et le second doit être 9

il n'y a pas de

Questions stupides

Q: Je peux donc utiliser dans mon CHECK tout ce que j'utiliserais dans une clause WHERE ?

R: Presque. Vous pouvez utiliser tous les conditionnels : AND, OR, IN, NOT, BETWEEN et les autres. Vous pouvez même les combiner, comme vous le voyez dans l'exemple ci-dessus. Mais vous ne pouvez pas utiliser de sous-requête.

Q: Mais si je ne peux pas les utiliser dans MySQL, que puis-je utiliser ?

R: Il n'y a pas de réponse évidente à cette question. Certaines personnes utilisent des déclencheurs (triggers), qui sont des requêtes qui s'exécutent si une certaine condition est remplie. Mais ils ne sont pas aussi simples que CHECK et dépassent la portée de ce livre.

Q: Que se passe-t-il si vous essayez d'insérer une valeur ne remplissant pas la condition du CHECK ?

R: Vous obtiendrez une erreur et rien ne sera inséré.

Q: En quoi est-ce utile ?

R: Cela permet de vérifier que les données saisies dans la table sont logiques. Vous ne vous retrouverez pas avec des valeurs mystères.

Le travail de Norbert devient fastidieux

Norbert travaille à faire correspondre les gens avec des offres d'emplois. Il remarque des récurrences. Il a beaucoup d'offres pour des concepteurs web et peu de candidats. Il a beaucoup d'écrivains techniques cherchant du travail, mais peu de postes.

Il exécute les mêmes requêtes tous les jours pour essayer de trouver des correspondances entre les gens et les emplois.

Vous ÊTES Norbert

Votre mission est de jouer le rôle de Norbert et d'écrire les requêtes qu'il écrit tous les jours. Écrivez une requête pour trouver tous les concepteurs web d'emploi_voulu ainsi que les infos pour les contacter. Écrivez-en une autre pour trouver les offres pour les écrivains techniques.

. .....

. .....

. .....

. .....

. .....

. .....

Vous ÊTES Norbert - SOLUTION

Votre mission est de jouer le rôle de Norbert et d'écrire les requêtes qu'il écrit tous les jours. Écrivez une requête pour trouver tous les concepteurs web d'emploi_voulu ainsi que les infos pour les contacter. Écrivez-en une autre pour trouver les offres pour les écrivains techniques.

```
SELECT mc.prenom, mc.nom, mc.telephone,
mc.email
FROM mes_contacts mc
NATURAL JOIN emploi_voulu ev
WHERE ev.profession = 'Concepteur web';
```

Greg met généralement une majuscule aux professions dans sa base de données.

```
SELECT profession, salaire,
description, code_postal
FROM offres_emplois
WHERE profession = 'Ecrivain technique';
```

Ce ne sont pas des requêtes difficiles, mais parce qu'il doit les retaper sans arrêt, Norbert va forcément faire des erreurs. Il doit trouver un moyen de sauvegarder les requêtes pour en voir le résultat une fois par jour sans devoir les retaper.

Il peut tout simplement les sauvegarder dans un fichier texte et les copier-coller. Quel est le problème ?

Les fichiers peuvent être réécrits ou modifiés.

Le fichier peut être accidentellement modifié ou supprimé. Il y a un bien meilleur moyen de sauvegarder ces requêtes **à l'intérieur** de la base de données elle-même. Nous pouvons en faire des **vues**.

Créer une vue

Créer une vue est très simple. Nous ajoutons une instruction `CREATE VIEW` à notre requête. Nous allons créer deux vues à partir des requêtes de Norbert :

```
CREATE VIEW concepteurs_web AS
SELECT mc.prenom, mc.nom, mc.telephone, mc.email
FROM mes_contacts mc
NATURAL JOIN emploi_voulu ev
WHERE ev.profession = 'Concepteur web';
```

Ceci aurait aussi pu être un INNER JOIN utilisant : ON mc.id_contact = ev.id_contact.

```
CREATE VIEW offres_ecrivain_tech AS
SELECT profession, salaire, description, code_postal
FROM offres_emplois
WHERE profession = 'Ecrivain technique';
```

MUSCLEZ VOS NEURONES

À ***votre*** avis, à quoi ressemble une instruction SQL utilisant une VUE ?

Voir vos vues

Prenez la vue `concepteurs_web` que nous venons de créer :

```
CREATE VIEW concepteurs_web AS
SELECT mc.prenom, mc.nom, mc.telephone, mc.email
FROM mes_contacts mc
NATURAL JOIN emploi_voulu ev
WHERE ev.profession = 'Concepteur web';
```

Rappelez-vous, on peut omettre le mot-clé AS.

Pour voir ce qu'une vue contient, nous la traitons simplement comme si c'était une table. Nous pouvons utiliser un `SELECT` :

```
SELECT * FROM concepteurs_web;
```

Voici le nom de notre vue.

Le résultat est :

prenom	nom	telephone	email
Amélie	Martinez	0612129872	am@unemail.com
Martine	Pommier	0612126948	martine@unemail.com
Laurent	Hertz	0611117888	laurent@unemail.com
Fred	Demelaere	0611117744	fd@unemail.com
Lou	Barr	0632328743	lou@oreilly.com

Etc, jusqu'à ce que toutes les lignes correspondant à « Concepteur web » soient affichées.

Ce que fait vraiment votre vue

Quand vous utilisez votre vue dans une requête, tout se passe en fait comme si c'était une sous-requête. Voici ce que le SELECT que nous venons d'utiliser avec notre vue demande de faire à SQL :

```
SELECT * FROM concepteurs_web;
```

Cela signifie : « Sélectionne tous les résultats de la sous-requête qui renvoie le prénom, nom, téléphone et email de toutes les personnes de mes_contacts qui cherchent un emploi de concepteur web ».

```
SELECT * FROM
(SELECT mc.prenom, mc.nom, mc.telephone, mc.email
FROM mes_contacts mc
NATURAL JOIN emploi_voulu ev
WHERE ev.profession = 'Concepteur web') AS concepteurs_web;
```

Voici ce que nous avons utilisé dans notre vue.

Nous donnons un alias à notre sous-requête pour que notre requête la traite comme une table.

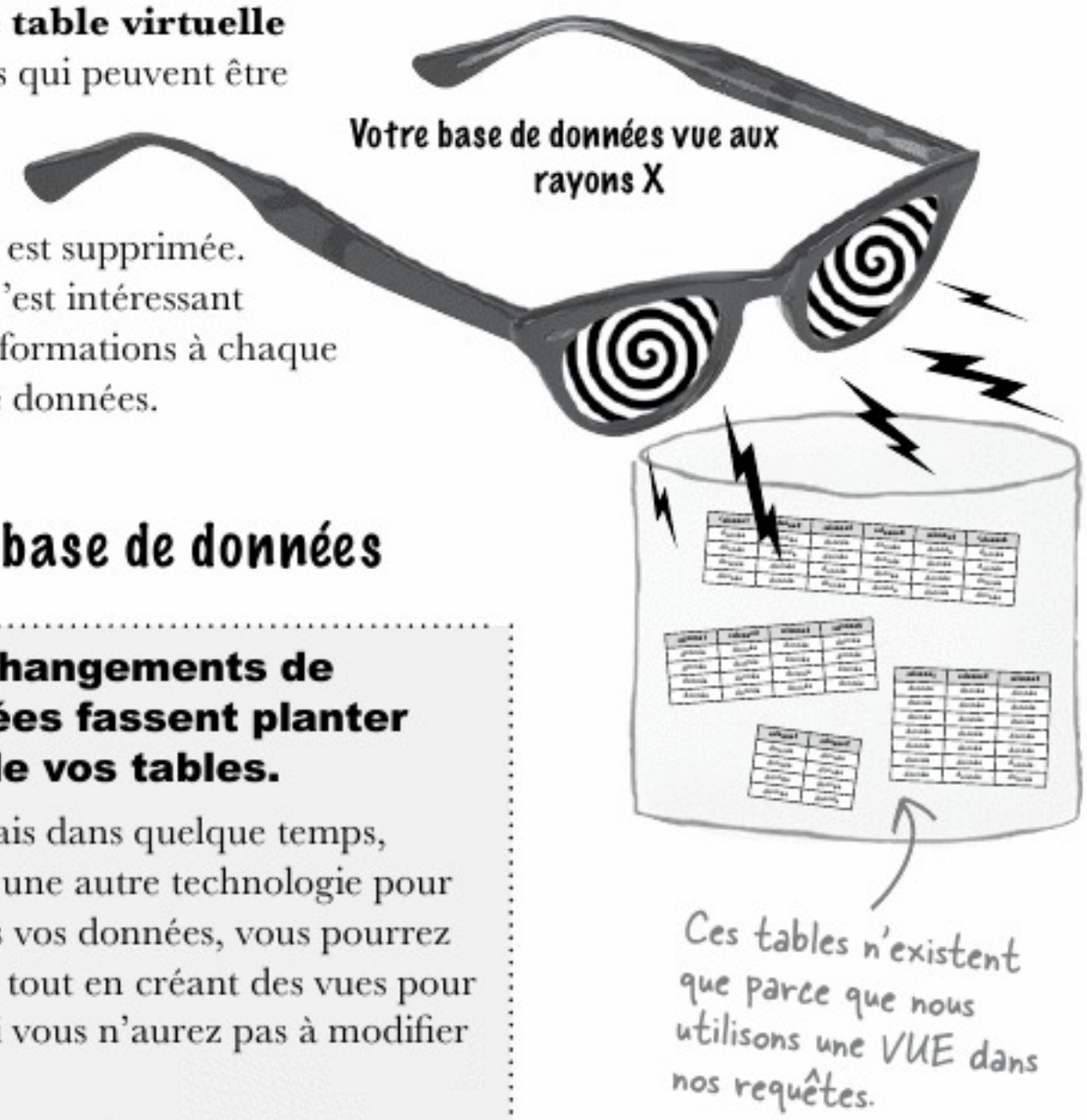
La clause FROM demande une table.

Et même si notre instruction SELECT renvoie une table virtuelle, il n'y a pas de moyen pour SQL de la récupérer sans cet alias.

Ce qu'est une vue

Essentiellement, une VUE est une table qui n'existe que quand vous utilisez la vue dans une requête. Elle est considérée comme une **table virtuelle** parce qu'elle agit comme une table et que les opérations qui peuvent être réalisées sur une table peuvent aussi l'être sur une vue.

Mais la table virtuelle ne reste pas dans la base de données. Elle est créée quand nous utilisons la vue, puis est supprimée. La VUE du même nom est la seule chose qui perdure. C'est intéressant parce qu'une vue va prendre en compte les nouvelles informations à chaque fois que de nouvelles lignes sont insérées dans la base de données.



Pourquoi les vues font du bien à votre base de données

1 **Vous pouvez empêcher que les changements de structure de votre base de données fassent planter les applications qui dépendent de vos tables.**

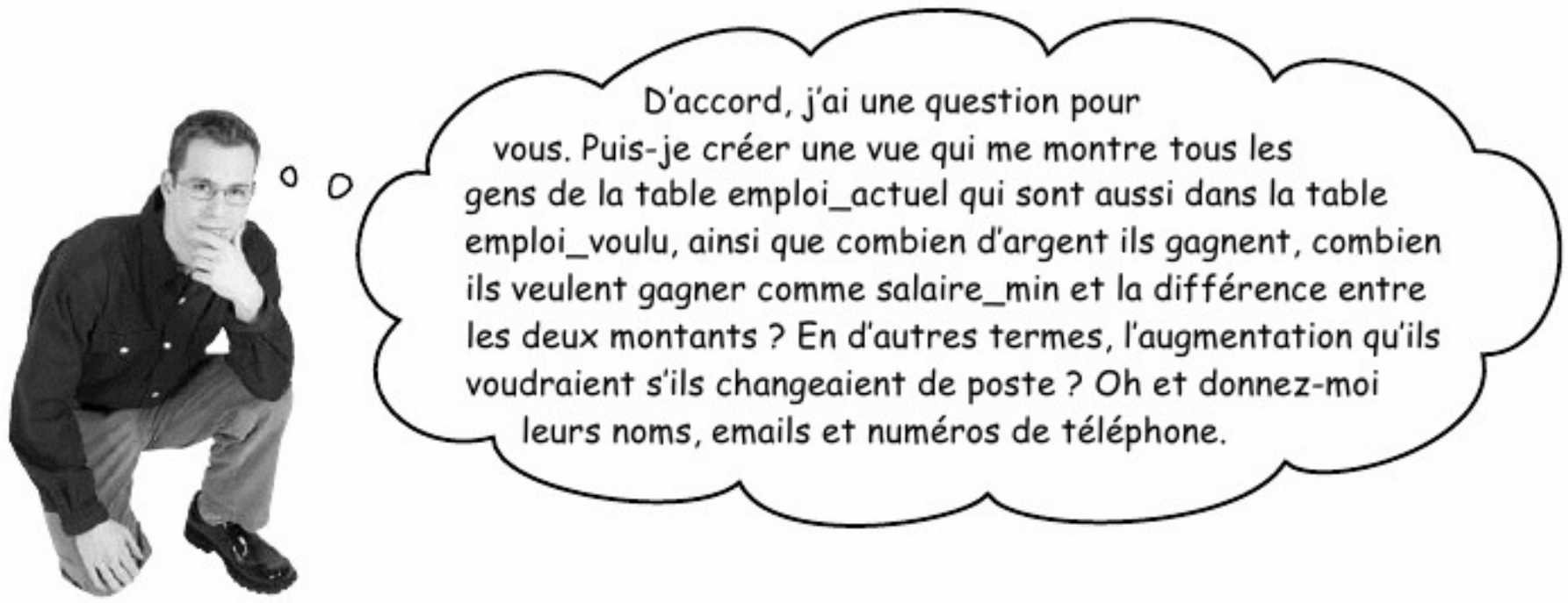
Nous n'en avons pas parlé dans ce livre, mais dans quelque temps, vous utiliserez votre connaissance d'SQL avec une autre technologie pour créer des applications. En créant des vues dans vos données, vous pourrez modifier la structure sous-jacente de vos tables tout en créant des vues pour imiter votre ancienne structure de tables. Ainsi vous n'aurez pas à modifier l'application utilisant vos données.

2 **Les vues vous simplifient la vie en transformant votre requête complexe en une simple commande.**

Vous ne devrez pas recréer sans arrêt des jointures et des sous-requêtes compliquées si vous pouvez créer une vue à la place. Votre vue dissimule la complexité de la requête sous-jacente. Et quand vous incorporerez votre SQL dans PHP ou un autre langage de programmation, votre vue sera bien plus facile à ajouter dans votre code. Vous utiliserez le code simplifié de la vue, pas la grande requête complexe pleine de jointures. La simplicité implique moins de risques pour la typo et votre code en sera d'autant plus facile à lire.

3 **Vous pouvez créer des vues pour cacher les informations non nécessaires à l'utilisateur.**

Considérez l'addition de tables contenant des informations de cartes bancaires dans `liste_greg`. Vous pouvez créer une vue pour indiquer que quelqu'un a une carte dans le fichier sans révéler les détails de cette carte. Vous pouvez permettre aux employés de ne voir que les informations dont ils ont besoin et garder les informations sensibles cachées.



C'est une tâche difficile, mais vous pouvez transformer en vue n'importe quelle requête SELECT que vous pouvez créer. Commencez par répondre aux questions ci-dessous, puis écrivez la requête de Norbert en tant que vue, appelée augmentations.

Quelles tables doivent figurer dans cette requête ?

. .....

Quelles colonnes de quelles tables peuvent être utilisées pour calculer l'augmentation ?

. .....

Comment pouvons-nous utiliser SQL pour créer une colonne appelée 'augmentation' dans nos résultats ?

. .....

Écrivez la requête de Norbert :

. .....

. .....

. .....

. .....

. .....

. .....

Indice : Essayez de l'écrire avec deux jointures sur trois tables !

C'est une tâche difficile, mais vous pouvez transformer en vue n'importe quelle requête SELECT que vous pouvez créer. Commencez par répondre aux questions ci-dessous, puis écrivez la requête de Norbert en tant que vue, appelée augmentations.

Quelles tables doivent figurer dans cette requête ?

emploi_actuel, emploi_voulu et mes_contacts

Quelles colonnes de quelles tables peuvent être utilisées pour calculer l'augmentation ?

La colonne salaire d'emploi_actuel et la colonne salaire_min d'emploi_voulu

Comment pouvons-nous utiliser SQL pour créer une colonne appelée 'augmentation' dans nos résultats ?

Soustraire le salaire actuel de salaire_min et lui donner un alias

Écrivez la requête de Norbert :

Ici, nous créons notre nouvelle vue appelée augmentations.

```
CREATE VIEW augmentations AS
SELECT mc.prenom, mc.nom, mc.email, mc.telephone, ea.salaire, ev.salaire_min,
ev.salaire_min - ea.salaire AS augmentation
FROM emploi_actuel ea
INNER JOIN emploi_voulu ev
INNER JOIN mes_contacts mc
WHERE ea.id_contact = ev.id_contact
AND ea.id_contact = mc.id_contact;
```

Après avoir créé la vue, le reste de la requête utilise deux INNER JOIN pour récupérer les données des trois tables. Nous utilisons aussi une petite correspondance pour créer notre nouvelle colonne 'augmentation'.

Ceci soustrait le salaire qu'ils veulent du salaire qu'ils ont actuellement et utilise un alias pour appeler le résultat 'augmentation'.

C'est une énorme requête, mais maintenant Norbert n'a plus qu'à taper

```
SELECT * FROM augmentations;
```

pour voir ses informations.

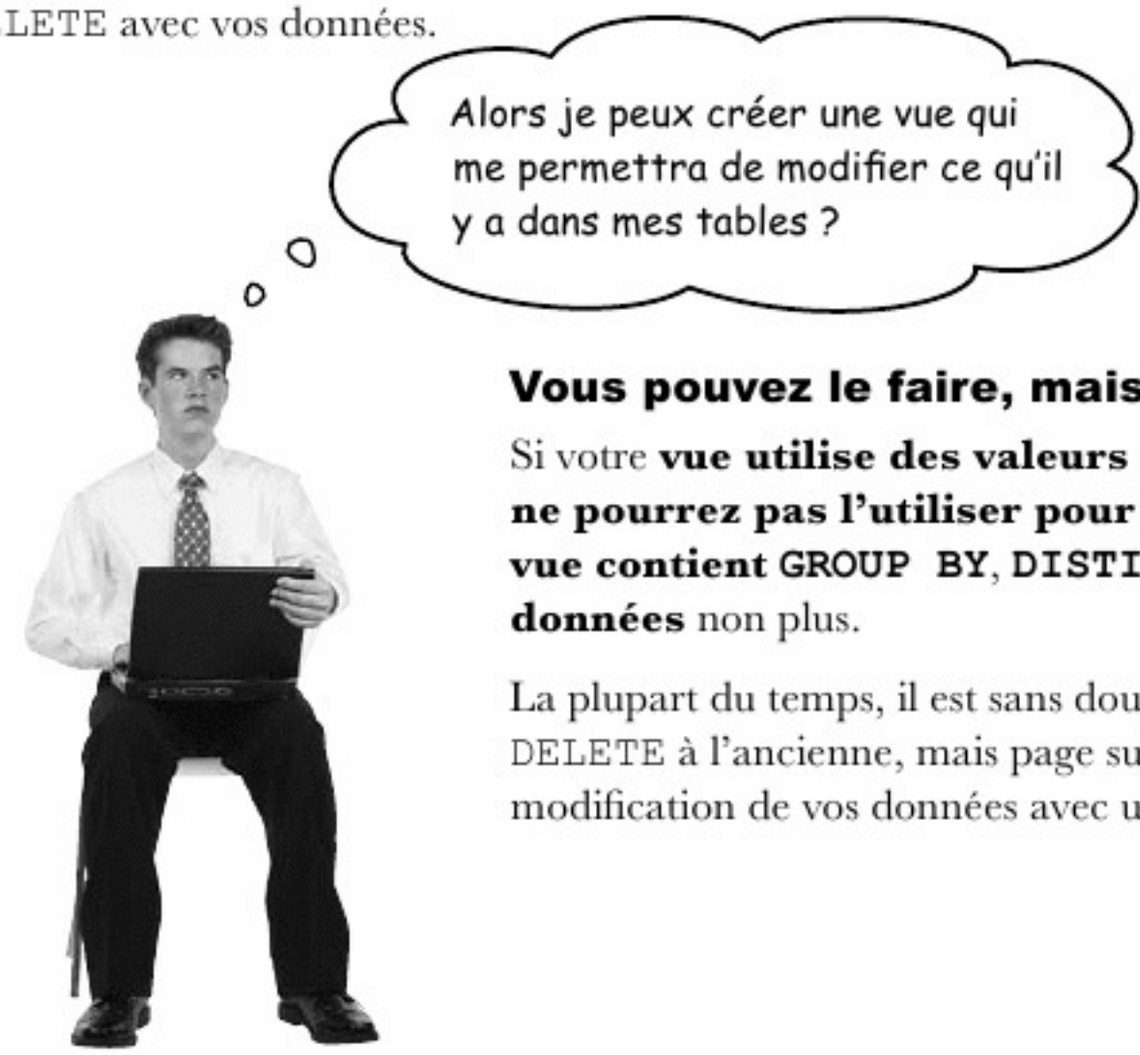
À vos crayons !

S'il exécute le SELECT de la page 470 en utilisant la nouvelle vue augmentations, comment Norbert peut-il trier ses résultats avec leurs noms dans l'ordre alphabétique ?

Réponse page 491.

Insert, update et delete avec des vues

Vous pouvez faire plus que de SELECTionner des informations de vos tables avec une vue. Dans certains cas, vous pouvez aussi utiliser UPDATE, INSERT et DELETE avec vos données.



Vous pouvez le faire, mais cela n'en vaut pas la peine.

Si votre **vue utilise des valeurs agrégées** (comme **SUM**, **COUNT** et **AVG**), **vous ne pourrez pas l'utiliser pour modifier des données**. De même, si votre **vue contient GROUP BY**, **DISTINCT** ou **HAVING**, cela **ne modifiera pas les données** non plus.

La plupart du temps, il est sans doute plus simple d'utiliser INSERT, UPDATE et DELETE à l'ancienne, mais page suivante, nous allons vous montrer un exemple de modification de vos données avec une vue.

Q: Pourriez-vous me donner quelques précisions sur HAVING ?

R: La clause HAVING spécifie un critère de recherche pour un groupe ou une fonction d'agrégation (SUM, AVG, etc). Elle est généralement utilisée avec GROUP BY. La clause WHERE permet d'écrire une restriction au niveau ligne alors que la clause HAVING permet d'écrire une restriction au niveau groupe.
Par exemple, la requête suivante liste les professions parmi les postes à pouvoir dont le salaire proposé moyen est supérieur à 10 000 euros :

```
SELECT profession, AVG(salaire) FROM offres_emplois
GROUP BY profession
HAVING AVG(salaire) > 10000;
```

Le secret est de prétendre qu'une vue est une vraie table

Créons une vue à partir d'une nouvelle table appelée `tirelire`. Cette table contient des pièces que nous collectionnons. Il y a une id pour chaque pièce ; une colonne qui indique le montant de la pièce et le pays où la pièce a été frappée.

```
CREATE TABLE tirelire
(
  id INT AUTO_INCREMENT NOT NULL PRIMARY KEY,
  piece DEC(3,2) NOT NULL,
  pays_piece CHAR(20)
)
```

Et voici les données actuellement dans la table `tirelire` :

id	piece	pays_piece
1	0.20	Portugal
2	1	Grèce
3	0.50	France
4	0.20	Belgique
5	0.20	Finlande
6	0.10	Irlande
7	0.20	Espagne
8	1	Italie
9	0.10	Allemagne
10	1	Belgique

Ecrivons une vue qui ne nous montre que les lignes contenant des pièces de 20 cents :

```
CREATE VIEW 20cents AS
SELECT * FROM tirelire
WHERE piece = 0.20;
```

À quoi ressemblera la table des résultats quand nous exécuterons cette requête ?

```
SELECT * FROM 20cents;
```

Essayez ceci chez vous. Créez la table tirelire et les vues 20cents et 10cents en utilisant les requêtes ci-dessous.

```
INSERT INTO tirelire (piece, pays_piece) VALUES (0.20, 'Portugal'), (1,
'Grèce'),(0.50, 'France'), (0.20, 'Belgique'),(0.20, 'Finlande'),(0.10,
'Irlande'),(0.20, 'Espagne'),(1, 'Italie'),(0.10, 'Allemagne'),(1, 'Belgique');

CREATE VIEW 20cents AS SELECT * FROM tirelire WHERE piece = 0.20;

CREATE VIEW 10cents AS SELECT * FROM tirelire WHERE piece = 0.10 WITH CHECK OPTION;
```

Essayez de comprendre l'effet de ceci à travers cet exercice.

Notez ce qui se passe quand vous exécutez ces requêtes INSERT, DELETE et UPDATE. À la fin de l'exercice, dessinez la table tirelire finale.

```
INSERT INTO 20cents (piece, pays_piece) VALUES (0.20, 'Luxembourg');
```

. .....

```
INSERT INTO 20cents (piece, pays_piece) VALUES (0.10, 'Autriche');
```

. .....

. .....

```
INSERT INTO 10cents (piece, pays_piece) VALUES (0.20, 'France');
```

. .....

. .....

. .....

```
DELETE FROM 20cents WHERE piece = 0.50 OR piece = 1 OR piece = 0.10;
```

. .....

. .....

```
UPDATE 20cents SET piece = 0.20 WHERE piece = 1;
```

. .....

. .....

. .....

Essayez ceci chez vous. Créez la table tirelire et les vues 20cents et 10cents en utilisant les requêtes ci-dessous.

```
INSERT INTO tirelire (piece, pays_piece) VALUES (0.20, 'Portugal'), (1,
'Grèce'),(0.50, 'France'), (0.20, 'Belgique'),(0.20, 'Finlande'),(0.10,
'Irlande'),(0.20, 'Espagne'),(1, 'Italie'),(0.10, 'Allemagne'),(1, 'Belgique');

CREATE VIEW 20cents AS SELECT * FROM tirelire WHERE piece = 0.20;

CREATE VIEW 10cents AS SELECT * FROM tirelire WHERE piece = 0.10 WITH CHECK OPTION;
```

Essayez de comprendre l'effet de ceci à travers cet exercice.

Notez ce qui se passe quand vous exécutez ces requêtes INSERT, DELETE et UPDATE. À la fin de l'exercice, dessinez la table tirelire finale.

```
INSERT INTO 20cents (piece, pays_piece) VALUES (0.20, 'Luxembourg');
```

Cette requête s'exécute correctement.

```
INSERT INTO 20cents (piece, pays_piece) VALUES (0.10, 'Autriche');
```

Ceci insère une nouvelle valeur dans la table, même si on pourrait penser que non à cause de la clause WHERE.

```
INSERT INTO 10cents (piece, pays_piece) VALUES (0.20, 'France');
```

Ceci vous donne une erreur à cause de la clause CHECK OPTION. Cela fait en sorte que les données entrées dans une vue soient confrontées à la clause WHERE avant d'être autorisées à être ajoutées.

```
DELETE FROM 20cents WHERE piece = 0.50 OR piece = 1 OR piece = 0.10;
```

Celle-ci ne fait rien du tout à la table parce qu'elle ne regarde que les résultats où piece = 0.20

```
UPDATE 20cents  SET piece = 0.20 WHERE piece = 1;
```

Celle-ci ne fait rien du tout à la table parce qu'aucune valeur de piece = 1 ne sont renvoyées par la vue 20cents.

La table finale ressemble à cela :

id	piece	pays_piece
1	0.20	Portugal
2	1	Grèce
3	0.50	France
4	0.20	Belgique
5	0.20	Finlande
6	0.10	Irlande
7	0.20	Espagne
8	1	Italie
9	0.10	Allemagne
10	1	Belgique
11	0.20	Luxembourg
12	0.10	Autriche

Vue avec CHECK OPTION

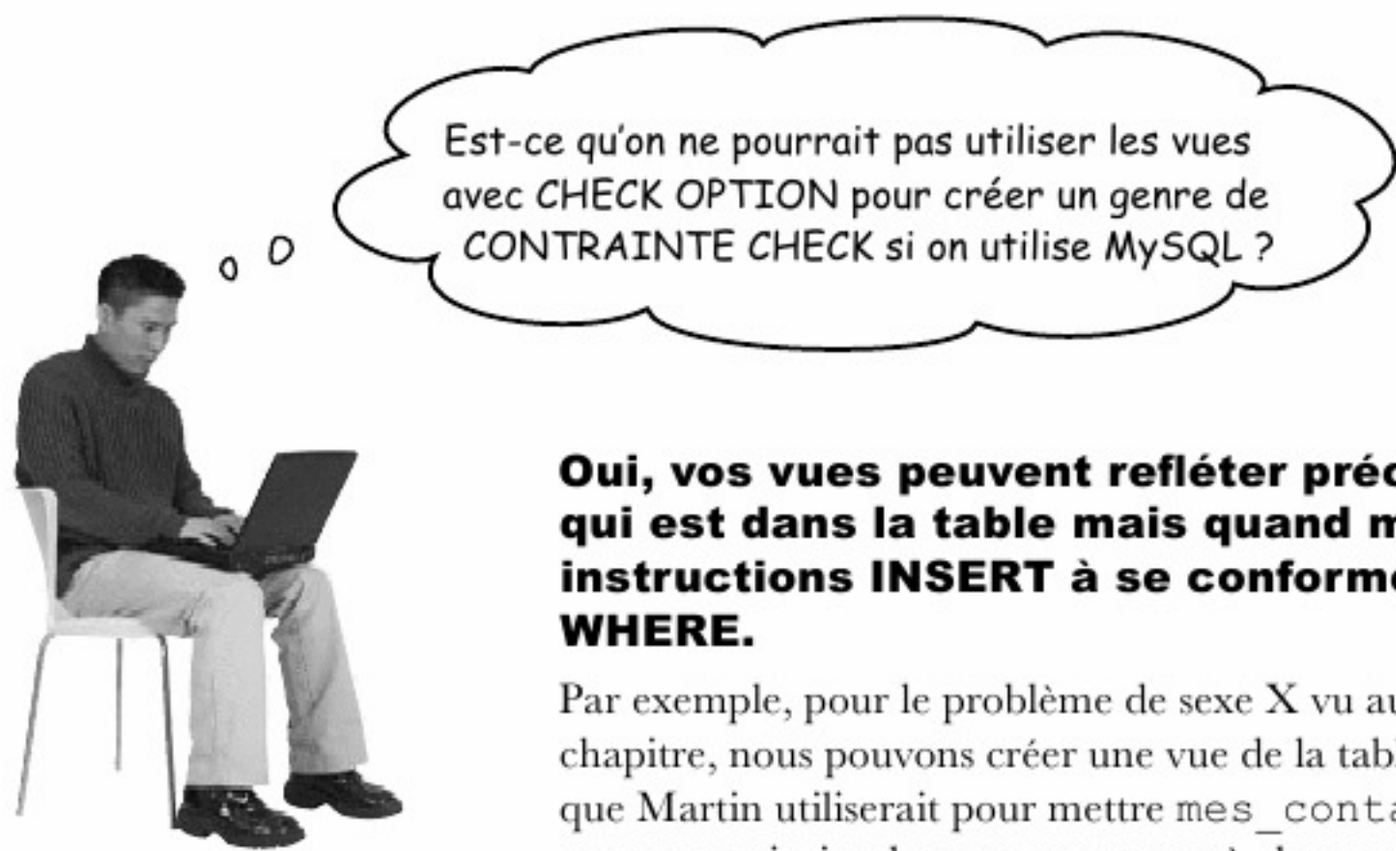
CHECK OPTION ajouté à votre vue demande au SGBDR de vérifier chaque instruction que vous entrez avec `INSERT` et `DELETE` pour voir si elle est autorisée par la clause `WHERE` de votre vue. Mais comment cette `CHECK OPTION` affecte-t-elle vos instructions `INSERT` et `UPDATE` ?

Quand vous avez utilisé `CHECK OPTION` dans l'exercice précédent, les données de votre `INSERT` ont été rejetées si elles ne correspondaient pas à la condition `WHERE` de la vue `10cents`. Vous obtiendrez également une erreur en utilisant `UPDATE` :

```
UPDATE 10cents SET piece = 5;
```

La condition `WHERE` de `10cents` n'est pas remplie par `5`, rien n'est donc modifié.

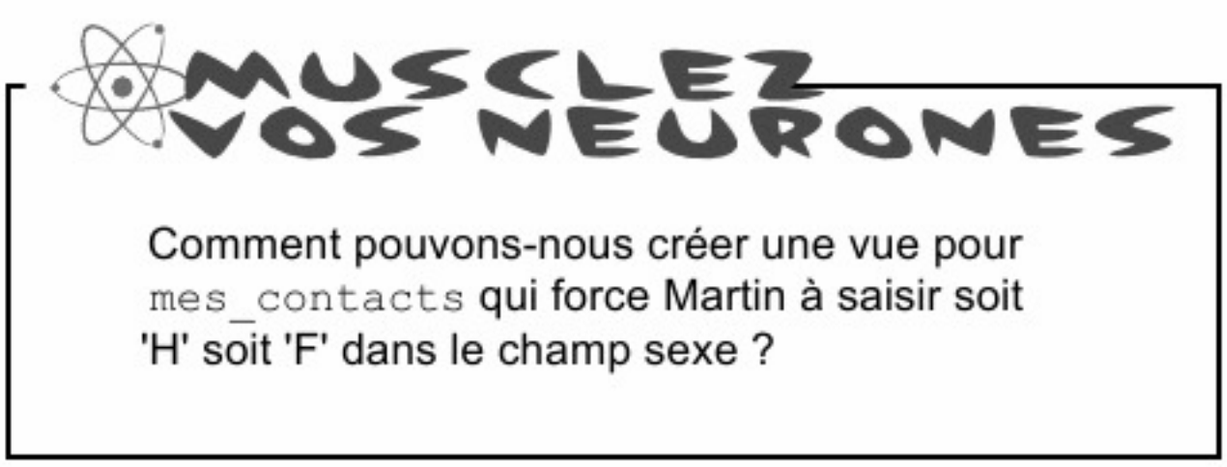
CHECK OPTION vérifie si chaque requête INSERT ou UPDATE que vous essayez de saisir est autorisée par la clause WHERE de votre vue.



Oui, vos vues peuvent refléter précisément ce qui est dans la table mais quand même forcer les instructions INSERT à se conformer aux clauses WHERE.

Par exemple, pour le problème de sexe X vu auparavant dans ce chapitre, nous pouvons créer une vue de la table `mes_contacts` que Martin utiliserait pour mettre `mes_contacts` à jour. Cela provoquerait simplement une erreur à chaque fois qu'il essaie de mettre X dans la colonne sexe.

Dans MySQL, vous pouvez simuler une CHECK CONSTRAINT avec CHECK OPTION



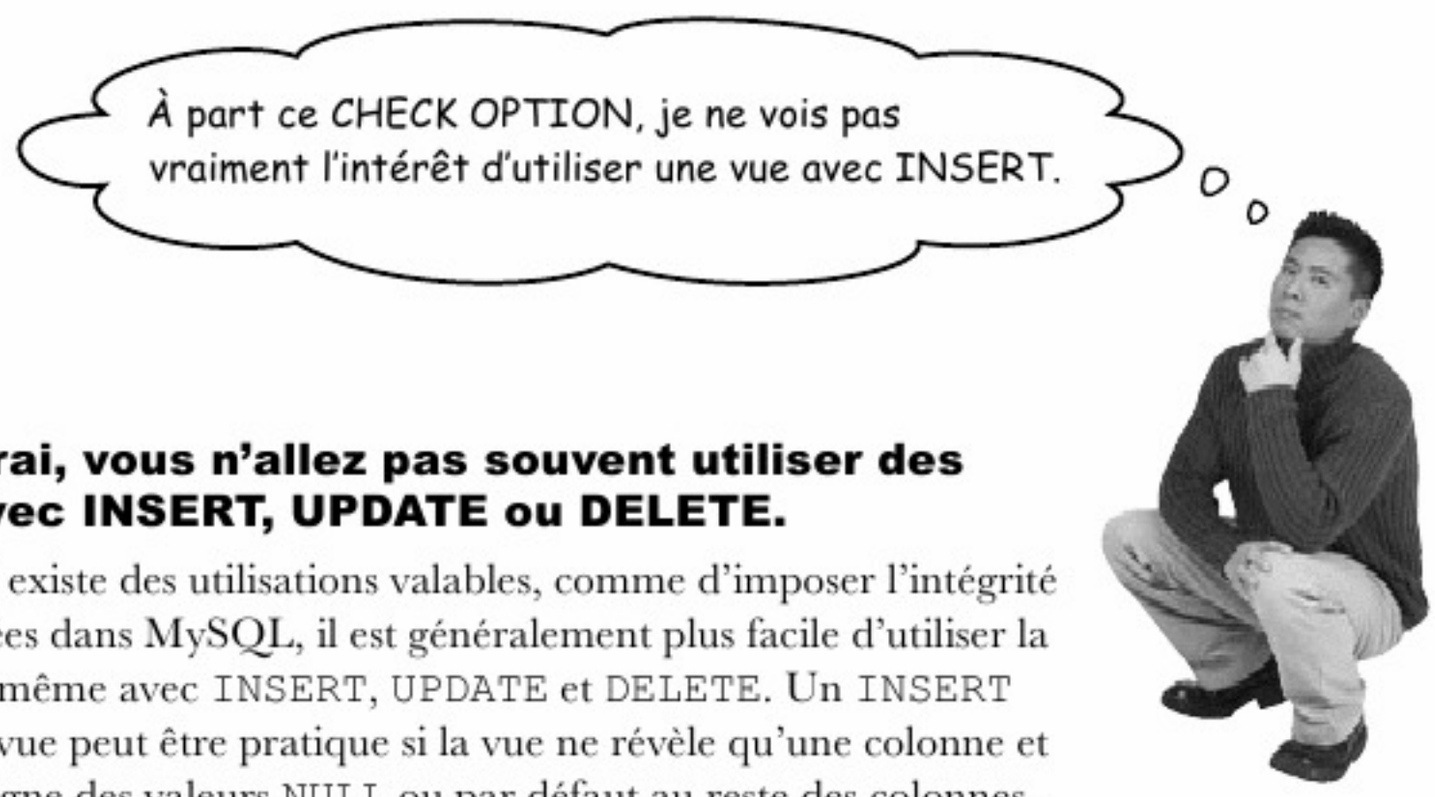
Votre vue est modifiable si...

Dans la table `tirelire`, les deux vues que nous avons créées peuvent être mises à jour. Une **vue modifiable** est une vue qui vous permet de modifier les tables sous-jacentes. L'important est qu'une vue modifiable inclue toutes les colonnes `NOT NULL` des tables auxquelles elle fait référence. Ainsi, quand vous insérez des données en utilisant une vue, vous pouvez être sûr que vous aurez une valeur pour toutes les colonnes pour lesquelles il vous en faut.

Essentiellement, cela signifie que `INSERT`, `UPDATE` et `DELETE` peuvent tous être utilisés avec les vues que nous créons. Tant que la vue renvoie toutes les colonnes de la table qui sont not null, la vue peut saisir les valeurs appropriées dans la table.

Il existe aussi des vues non-modifiables. Une **vue non-modifiable** est une vue qui n'inclut pas toutes les colonnes `NOT NULL`. À part la créer et la supprimer, la seule chose que vous pouvez faire avec une vue non-modifiable est d'y `SELECTionner` quelque chose.

Un vue modifiable inclut toutes les colonnes NOT NULL des tables auxquelles elle fait référence.



C'est vrai, vous n'allez pas souvent utiliser des vues avec INSERT, UPDATE ou DELETE.

Même s'il existe des utilisations valables, comme d'imposer l'intégrité des données dans MySQL, il est généralement plus facile d'utiliser la table elle-même avec `INSERT`, `UPDATE` et `DELETE`. Un `INSERT` dans une vue peut être pratique si la vue ne révèle qu'une colonne et qu'on assigne des valeurs `NULL` ou par défaut au reste des colonnes. Dans ce cas, `INSERT` peut être logique. Vous pouvez aussi ajouter une clause `WHERE` à votre vue pour restreindre ce que l'on peut insérer, pour vous aider à simuler une contrainte `CHECK` dans MySQL.

Pour compliquer encore plus les choses, vous ne pouvez modifier que les vues qui ne contiennent pas d'opérateurs aggrégés comme `SUM`, `COUNT` et `AVG` ou d'opérateurs comme `BETWEEN`, `HAVING`, `IN` et `NOT IN`.

Quand vous en avez terminé avec votre vue

Si vous n'avez plus besoin d'une de vos vues, nettoyez-la en utilisant une instruction `DROP VIEW`. C'est aussi simple que :

```
DROP VIEW 10cents;
```

il n'y a pas de

Questions stupides

Q: Existe-t-il un moyen de voir quelles vues on a créé ?

R: Les vues apparaissent de la même façon que les tables dans votre base de données. Vous pouvez utiliser la commande `SHOW TABLES` pour voir toutes les vues et les tables. Et tout comme pour une table, vous pouvez utiliser `DESC` pour voir la structure d'une vue.

Q: Que se passe-t-il si je supprime une table qui a une vue ?

R: Cela dépend. Certains SGBDR vous permettront d'utiliser la vue et ne renverront aucune donnée. MySQL ne vous permettra pas de supprimer une vue si la table sur laquelle elle est basée n'existe pas, même si vous **pouvez** supprimer une table participant à une vue. D'autres SGBDR se comportent différemment. Il peut être utile de faire quelques expériences avec le vôtre pour voir ce qui se passe. En général, il vaut mieux supprimer la vue avant de supprimer une table sur laquelle elle s'appuie.

Q: Je vois combien les contraintes CHECK et les vues sont utiles quand plus d'une personne essaient de faire des choses dans la base de données. Mais qu'arrive-t-il si deux personnes essaient de modifier la même colonne en même temps ?

R: Pour cela, nous devons parler des transactions. Mais d'abord, Mme Roquefort a besoin de retirer de l'argent.

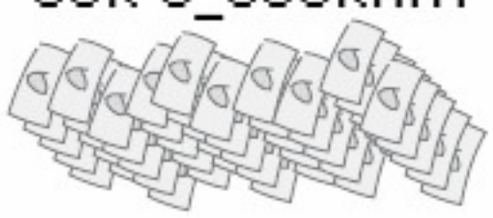
Les contraintes CHECK et les vues aident toutes deux à garder le contrôle quand vous avez plusieurs utilisateurs.

Quand il arrive du mal à de bonnes bases de données

Mme Roquefort veut transférer 1 000 bouzoufs de son compte courant vers son compte d'épargne. Elle se dirige vers le distributeur...

Elle vérifie le solde de son compte courant et de son compte d'épargne.

1000 BOUZOUFS
SUR C_COURANT



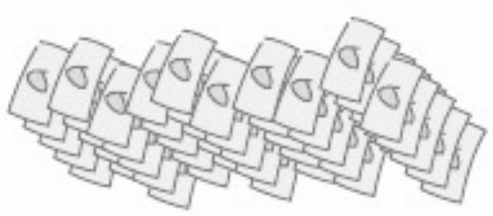
30 BOUZOUFS
SUR C_EPARGNE

Elle sélectionne.

TRANSFERER 1000 BOUZOUFS
DE C_COURANT VERS C_EPARGNE

Elle appuie sur le bouton.

C_COURANT → C_EPARGNE



Le distributeur fait bip et l'écran devient noir.

Le courant s'est coupé.

Le courant revient.

Elle vérifie le solde de ses comptes.

0 BOUZOUFS
SUR C_COURANT

30 BOUZOUFS
SUR C_EPARGNE

Où, mais où sont donc passés les bouzoufs de Mme Roquefort ?

Ce qui s'est passé dans le distributeur

Distributeur : LA LA LA LA LA

Distributeur : HE, C'EST MME MARTHE ROQUEFORT. BONJOUR MME MARTHE ROQUEFORT ! (ID_COMPTE 38221)

Mme Roquefort : Dites-moi combien j'ai d'argent.

Distributeur : *Refléchit*
(SELECT SOLDE FROM C_COURANT WHERE ID_COMPTE 38221;
SELECT SOLDE FROM C_EPARGNE WHERE ID_COMPTE 38221;)
CELA FAIT 1000 SUR C_COURANT, 30 SUR C_EPARGNE

Mme Roquefort : Transférez ces 1000 bouzoufs de mon compte courant à mon compte d'épargne.

Distributeur : C'EST UNE COMMANDE DIFFICILE, MME ROQUEFORT, MAIS VOILA :

(VERIFIE QUE SOLDE > 1000, DONC ELLE A ASSEZ D'ARGENT)
(ENLEVE 1000 DE C_COURANT)
(INSERT BIIIIIIIIP.....

C'est la que le courant s'est coupé.

Distributeur :

Distributeur :

Distributeur : Z Z Z Z Z Z Z Z Z

Distributeur : WAOOOH

Distributeur : HE, C'EST MME MARTHE ROQUEFORT. BONJOUR MME MARTHE ROQUEFORT ! (ID_COMPTE 38221)

Mme Roquefort: Dites-moi combien j'ai d'argent.

Distributeur : *Réfléchit*
(SELECT SOLDE FROM C_COURANT WHERE ID_COMPTE 38221;
SELECT SOLDE FROM C_EPARGNE WHERE ID_COMPTE 38221;)
CELA FAIT 0 SUR C_COURANT, 30 SUR C_EPARGNE

Distributeur : OUILLE ! C'EST MON ECRAN QUE VOUS MARTELEZ COMME CA. AU REVOIR MME MARTHE ROQUEFORT !

MUSCLEZ VOS NEURONES

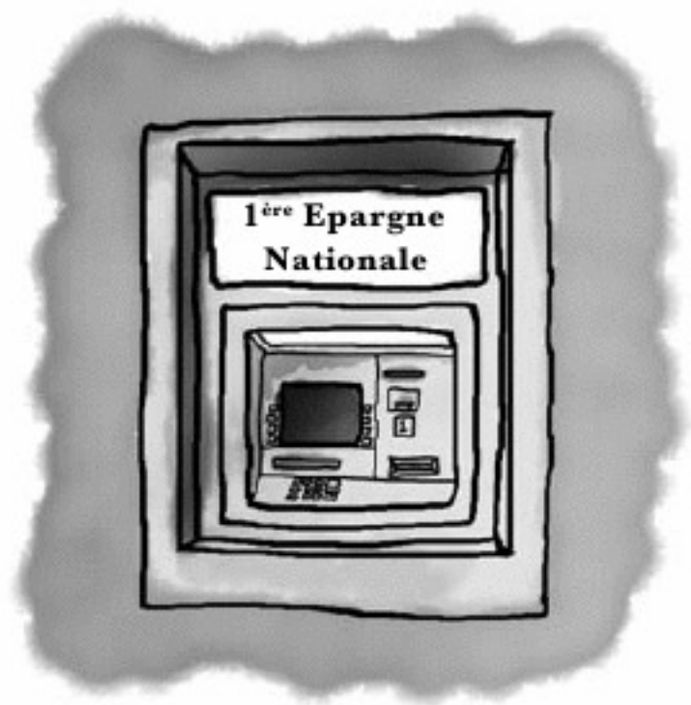
Comment aurions-nous pu empêcher le distributeur d'oublier la partie INSERT de la transaction de Mme Roquefort ?

Pendant ce temps-là, de l'autre côté de la ville…

D'autres problèmes au distributeur

Jean et Marie ont un compte commun. Un vendredi, ils se retrouvent à deux distributeurs différents au même moment. Ils essaient chacun de retirer 300 bouzoufs.

Voici la base de données gardant la trace de l'argent sur le compte joint de Marie et Jean.



Distributeur : OH, ENCORE TOI, JEAN. TU CROIS QUE JE SUIS PLEIN D'ARGENT OU QUOI ?

Jean : Quel est mon solde ?

Distributeur : *Réfléchit* (SELECT SOLDE FROM C_COURANT;) 350 BOUZOUFS

Jean : Donnez-moi 300 bouzoufs

Distributeur : TU PENSES QUE JE NE SERS QU'A CA, DONNER DE L'ARGENT. TU M'UTILISES ET APRES TU M'IGNORES.

(VERIFIE QUE SOLDE > 300, IL A ASSEZ D'ARGENT)

(RETIRE 300 DE C_COURANT)

(SOUSTRAIT 300 DE SOLDE)

350 bouzoufs

50 bouzoufs

Jean prend l'argent et s'en va.

Distributeur : TU N'APPELLES JAMAIS, TU N'ECRIS JAMAIS. AU REVOIR JEAN.

C'est là que les choses ont mal tourné.

Distributeur : MARIE, SALUT TOI.

Marie : Quel est mon solde ?

Distributeur : *Réfléchit* (SELECT SOLDE FROM C_COURANT;)

350 BOUZOUFS

DRING DRING

Marie fouille dans son sac à la recherche de son téléphone portable.

Marie : Donnez-moi 300 bouzoufs.

Distributeur : BEN TIENS DONC

350 bouzoufs

(VERIFIE QUE SOLDE > 300, ELLE A ASSEZ D'ARGENT)

(RETIRE 300 DE C_COURANT)

-250 bouzoufs

(SOUSTRAIT 300 DE SOLDE)

Distributeur : VOUS AVEZ UN TRES GROS DECOUVERT

Ne serait-ce pas merveilleux si on pouvait exécuter une série d'instructions SQL groupées, toutes en une fois, et si quand quelque chose tournait mal on pouvait revenir en arrière comme si cela n'avait jamais été exécuté ? Mais ce n'est qu'un rêve...

Ce n'est pas un rêve, c'est une transaction

Une **transaction** est un ensemble d'instructions SQL qui accomplissent une unité de travail. Dans le cas de Mme Roquefort, une transaction comporterait toutes les instructions SQL nécessaires pour transférer l'argent de son compte courant vers son compte d'épargne :

Si le solde du compte courant >= 1000

Soustraire 1000 du solde du compte courant

Ajouter 1000 au solde du compte d'épargne

Ces trois actions forment une seule unité de travail. Ici c'est une transaction.

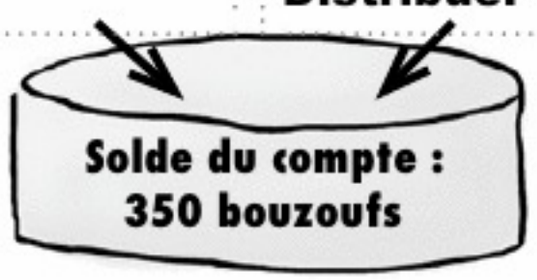
Jean et Marie essayaient tous les deux de réaliser ***la même transaction*** en même temps :

Jean et Marie essayaient tous les deux de retirer 300 bouzoufs en même temps.

Si le solde du compte courant >= 300	**Si le solde du compte courant >= 300**
Soustraire 300 du solde du compte courant	**Soustraire 300 du solde du compte courant**
Distribuer 300 bouzoufs	**Distribuer 300 bouzoufs**

La transaction de Jean au distributeur de la Banque de gauche

La transaction de Marie au distributeur de la 1ère Epargne Nationale.

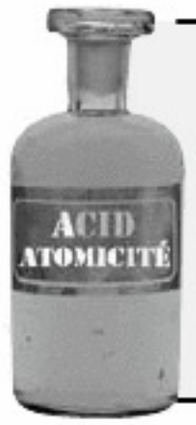


Dans le cas de Jean et Marie, le distributeur de la 1ère Epargne Nationale n'aurait pas dû être autorisé à toucher au compte, même pour demander le solde, jusqu'à ce que le distributeur de la Bangue de gauche ait terminé la transaction, la débloquant ainsi.

Pendant une transaction, si toutes les étapes ne peuvent pas être achevées sans interférence, aucune ne doit l'être.

Le test classique de l'ACID

Pour vous aider à décider quelles étapes peuvent être considérées comme une transaction dans votre SQL, rappelez-vous de l'acronyme **ACID**. Quatre caractéristiques doivent être vraies pour que l'on puisse appeler un ensemble d'instructions SQL une transaction :

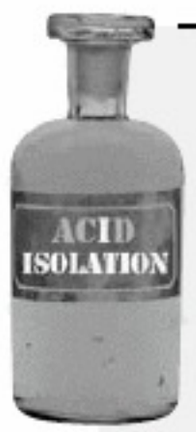


ACID : ATOMICITÉ

Toutes les parties de la transaction doivent être exécutées ou aucune d'elles ne le sera. Vous ne pouvez pas exécuter une partie d'une transaction. Les bouzoufs de Mme Roquefort se sont évaporés dans la panne de courant parce que seule une partie de la transaction a eu lieu.

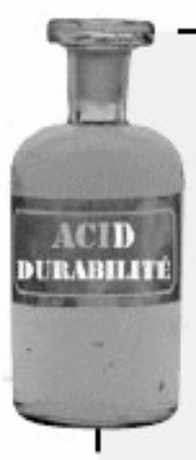
ACID : COHÉRENCE

Une transaction complète laisse la base de données dans un état cohérent à la fin de la transaction. À la fin des deux transactions de bouzoufs, l'argent est rééquilibré. Dans le premier cas il a été transféré vers le compte d'épargne ; dans le second il a été converti en liquide. Mais aucun bouzouf ne manque.



ACID : ISOLATION

Isolation signifie que toute transaction a une vue cohérente de la base de données indépendamment des autres transactions se déroulant au même moment. C'est ce qui a posé porblème pour Jean et Marie : le distributeur de Marie pouvait voir le solde alors que le distributeur de Jean effectuait la transaction. Elle n'aurait pas dû pouvoir voir le solde, ou elle aurait dû avoir un message du genre « transaction en cours ».



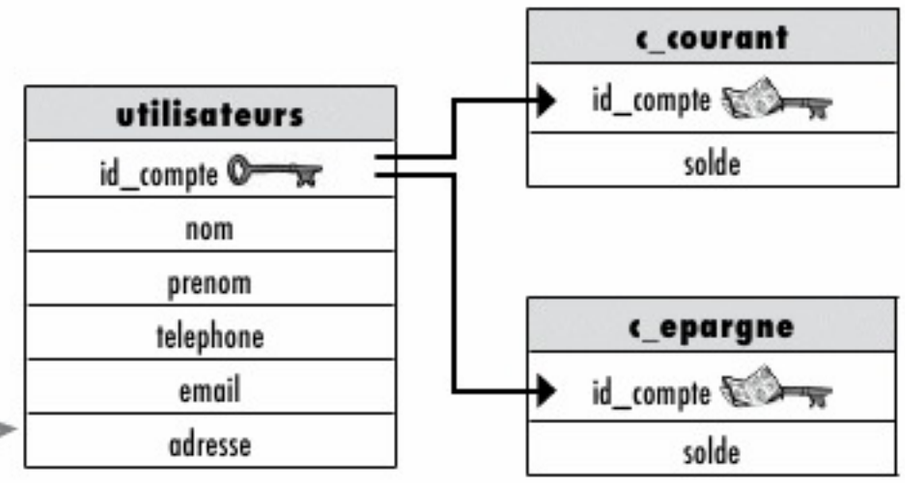
ACID : DURABILITÉ

Après la transaction, la base de données doit sauvegarder les données correctement et les protéger des pannes de courant et autres menaces. Généralement on gère cela en sauvegardant les enregistrements des transactions ailleurs que dans la base de données principale. Si un enregistrement de la transaction de Mme Roquefort avait été conservé quelque part, alors elle aurait pu récupérer ses 1 000 bouzoufs.

SQL vous aide à gérer vos transactions

Prenons une base de données de banque très simple. Notre base de données comporte une table des utilisateurs de comptes, une table pour le compte courant et une table pour le compte d'épargne :

Il y a probablement beaucoup plus de colonnes ici, mais vous saisissez l'idée.



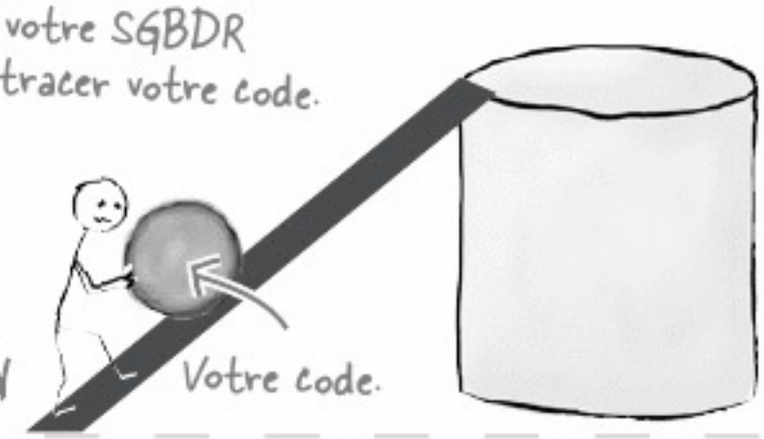
Nous avons trois outils de transaction SQL pour nous protéger :

START TRANSACTION;

Ceci trace ce que fait votre SQL.

C'est là que votre SGBDR commence à tracer votre code.

START TRANSACTION ***garde la trace de tout le SQL*** qui suit jusqu'à ce que vous saisissiez soit `COMMIT` soit `ROLLBACK`.

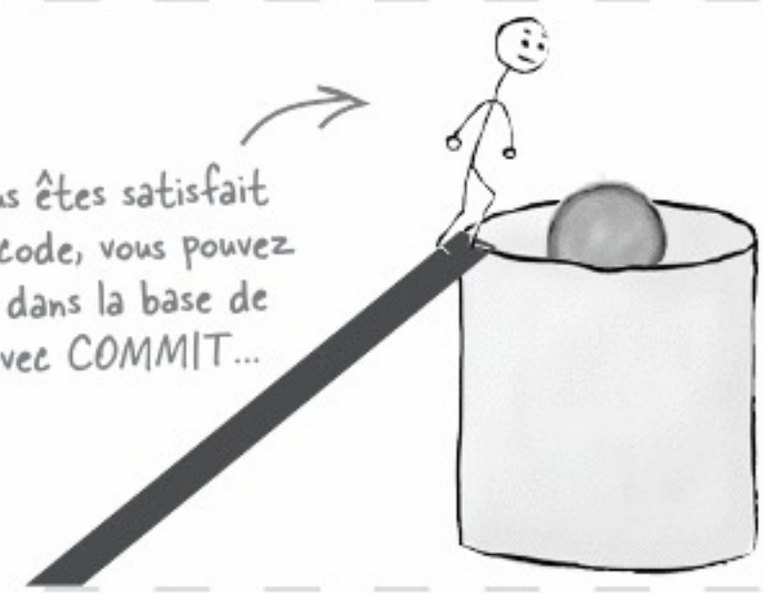


COMMIT;

Ceci valide tout votre code une fois que vous en êtes satisfait.

Quand vous êtes satisfait de votre code, vous pouvez le valider dans la base de données avec COMMIT...

Si toutes vos instructions sont en place et que tout semble bon, tapez **COMMIT** pour ***les rendre permanentes***.

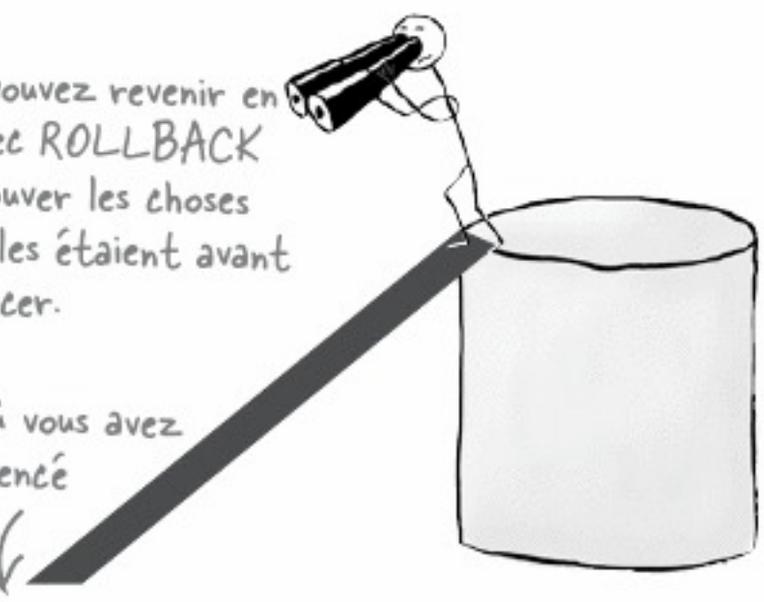


ROLLBACK;

Ceci vous ramène directement là où vous étiez avant de commencer la transaction.

... ou vous pouvez revenir en arrière avec ROLLBACK pour retrouver les choses telles qu'elles étaient avant de commencer.

Si quelque chose ne va pas, **ROLLBACK** ***annule tout*** et vous ramène à la situation ***avant*** que vous ne tapiez `START TRANSACTION`.



Aucun changement ne se produira dans la bdd à moins de taper COMMIT

Ce qui aurait dû se passer dans le distributeur

Distributeur : LA LA LA LA LA

Distributeur : HE, C'EST MME MARTHE ROQUEFORT. BONJOUR MME MARTHE ROQUEFORT ! (ID_COMPTE 38221)

Mme Roquefort : Dites-moi combien j'ai d'argent.

Distributeur : *Réfléchit*
(SELECT SOLDE FROM C_COURANT WHERE ID_COMPTE 38221;
SELECT SOLDE FROM C_EPARGNE WHERE ID_COMPTE 38221;)
CELA FAIT 1000 SUR C_COURANT, 30 SUR C_EPARGNE

Mme Roquefort : Transférez ces 1000 bouzoufs de mon compte courant à mon compte d'épargne.

Distributeur : C'EST UNE COMMANDE DIFFICILE, MME ROQUEFORT, MAIS VOILA :
(START TRANSACTION;
SELECT SOLDE FROM C_COURANT WHERE ID_COMPTE 38221;)

Distributeur : ELLE A 1000 SUR C_COURANT, DONC JE CONTINUE.

Distributeur : (UPDATE C_COURANT SET SOLDE SOLDE-1000 WHERE ID_COMPTE 38221;)

INSERT BIIIIIIIIIP.....

DISTRIBUTEUR SUR GENERATEUR DE SECOURS : ROLLBACK

C'est là qu'a eu lieu la panne de courant.

Distributeur :

Distributeur :

Distributeur : Z Z Z Z Z Z Z Z

Distributeur : WAOOOH

Distributeur : HE, C'EST MME MARTHE ROQUEFORT. BONJOUR MME MARTHE ROQUEFORT ! (ID_COMPTE 38221)

Mme Roquefort : Dites-moi combien j'ai d'argent.

Grâce à ROLLBACK, l'instruction COMMIT n'a jamais été entrée, rien n'a donc changé.

Distributeur : *Réfléchit*
(SELECT SOLDE FROM C_COURANT WHERE ID_COMPTE 38221;
SELECT SOLDE FROM C_EPARGNE WHERE ID_COMPTE 38221;)
CELA FAIT 10000 SUR C_COURANT, 30 SUR C_EPARGNE

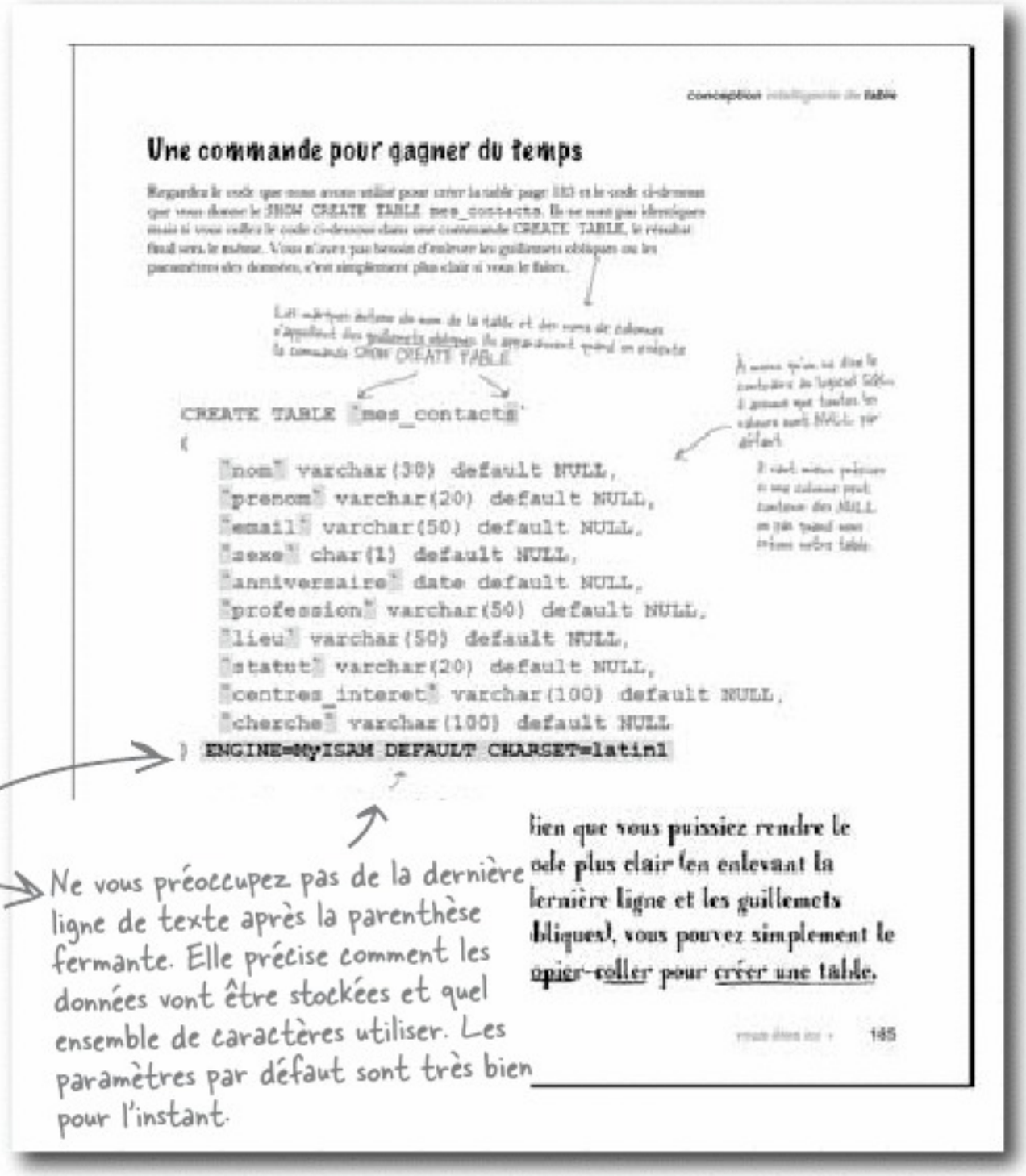
Comment faire fonctionner des transactions avec MySQL

Avant de pouvoir utiliser une transaction avec MySQL, vous devez utiliser le bon **moteur de stockage**. Le moteur de stockage est la structure qui stocke en coulisse toutes les données et les structures de votre base de données. Certains types permettent les transactions ; d'autres non.

Rappelez-vous du chapitre 4 quand vous avez vu :

```
SHOW CREATE TABLE mes_contacts;
```

Cette fois le moteur de stockage est important.



Une commande pour gagner du temps

```
CREATE TABLE `mes_contacts`
(
  `nom` varchar(30) default NULL,
  `prenom` varchar(20) default NULL,
  `email` varchar(50) default NULL,
  `sexe` char(1) default NULL,
  `anniversaire` date default NULL,
  `profession` varchar(50) default NULL,
  `lieu` varchar(50) default NULL,
  `statut` varchar(20) default NULL,
  `centres_interet` varchar(100) default NULL,
  `cherche` varchar(100) default NULL
) ENGINE=MyISAM DEFAULT CHARSET=latin1
```

Ne vous préoccupez pas de la dernière ligne de texte après la parenthèse fermante. Elle précise comment les données vont être stockées et quel ensemble de caractères utiliser. Les paramètres par défaut sont très bien pour l'instant.

lien que vous puissiez rendre le code plus clair (en enlevant la dernière ligne et les guillemets obliques), vous pouvez simplement le copier-coller pour créer une table.

Vous devez vérifier que votre moteur de stockage est soit BDB soit InnoDB, les deux choix qui permettent les transactions.

InnoDB et BDB sont deux moyens qu'a votre SGBDR pour stocker des données en coulisse.

Ils s'appellent des moteurs de stockage et utiliser l'un ou l'autre de ces types permet d'utiliser les transactions. Consultez une référence pour en savoir plus sur les différences entre les moteurs de stockage qu'offre MySQL.

Pour notre objectif, vous pouvez choisir celui que vous voulez. Pour changer de moteur, utilisez cette syntaxe :

```
ALTER TABLE votre_table TYPE = InnoDB;
```

Maintenant essayez-le vous-même

Supposez que nous ayons remplacé toutes les pièces d'un euro de notre tirelire par des pièces de deux euros.

Essayez le code ci-dessous sur la table `tirelire` que nous avons créée plus tôt dans ce chapitre. La première fois, nous allons utiliser `ROLLBACK` parce que nous avons décidé de ne pas effectuer nos modifications :

```
START TRANSACTION;
SELECT * FROM tirelire;
UPDATE tirelire SET piece = 2 WHERE piece= 1;
SELECT * FROM tirelire;
ROLLBACK;
SELECT * FROM tirelire;
```

Maintenant vous voyez les modifications...

Nous avons changé d'avis.

...et maintenant non.

La deuxième fois nous allons utiliser `COMMIT` parce nous sommes d'accord avec les modifications :

```
START TRANSACTION;
SELECT * FROM tirelire;
UPDATE tirelire SET piece = 2 WHERE piece= 1;
SELECT * FROM tirelire;
COMMIT;
SELECT * FROM tirelire;
```

Maintenant vous voyez les modifications...

Rend les modifications permantentes.

...et maintenant vous les voyez toujours.

À vos crayons !

Complétez le contenu de la tirelire après ces transactions. La voici comme elle est maintenant :

tirelire

id	piece	pays_piece
1	0.20	Portugal
2	1	Grèce
3	0.50	France
4	0.20	Belgique

```
START TRANSACTION;
UPDATE tirelire SET piece = 0.20 WHERE piece = 1
AND pays_piece < 'France';
COMMIT;
```

id	piece	pays_piece
1		
2		
3		
4		

```
START TRANSACTION;
UPDATE tirelire SET piece = 0.50 WHERE piece = 0.20;
ROLLBACK;
```

id	piece	pays_piece
1		
2		
3		
4		

```
START TRANSACTION;
UPDATE tirelire SET piece = 0.20 WHERE piece = 0.50
AND pays_piece < 'Portugal';
ROLLBACK;
```

id	piece	pays_piece
1		
2		
3		
4		

```
START TRANSACTION;
UPDATE tirelire SET piece = 0.10 WHERE piece = 0.20
AND pays_piece > 'Espagne';
COMMIT;
```

id	piece	pays_piece
1		
2		
3		
4		

```
START TRANSACTION;
UPDATE tirelire SET piece = 1 WHERE piece = 0.50
AND pays_piece < 'Grèce';
COMMIT;
```

id	piece	pays_piece
1		
2		
3		
4		

→ Réponses page 492.

il n'y a pas de
Questions stupides

Q: Est-il obligatoire de commencer par START TRANSACTION ou est-ce que COMMIT et ROLLBACK fonctionnent sans cela ?

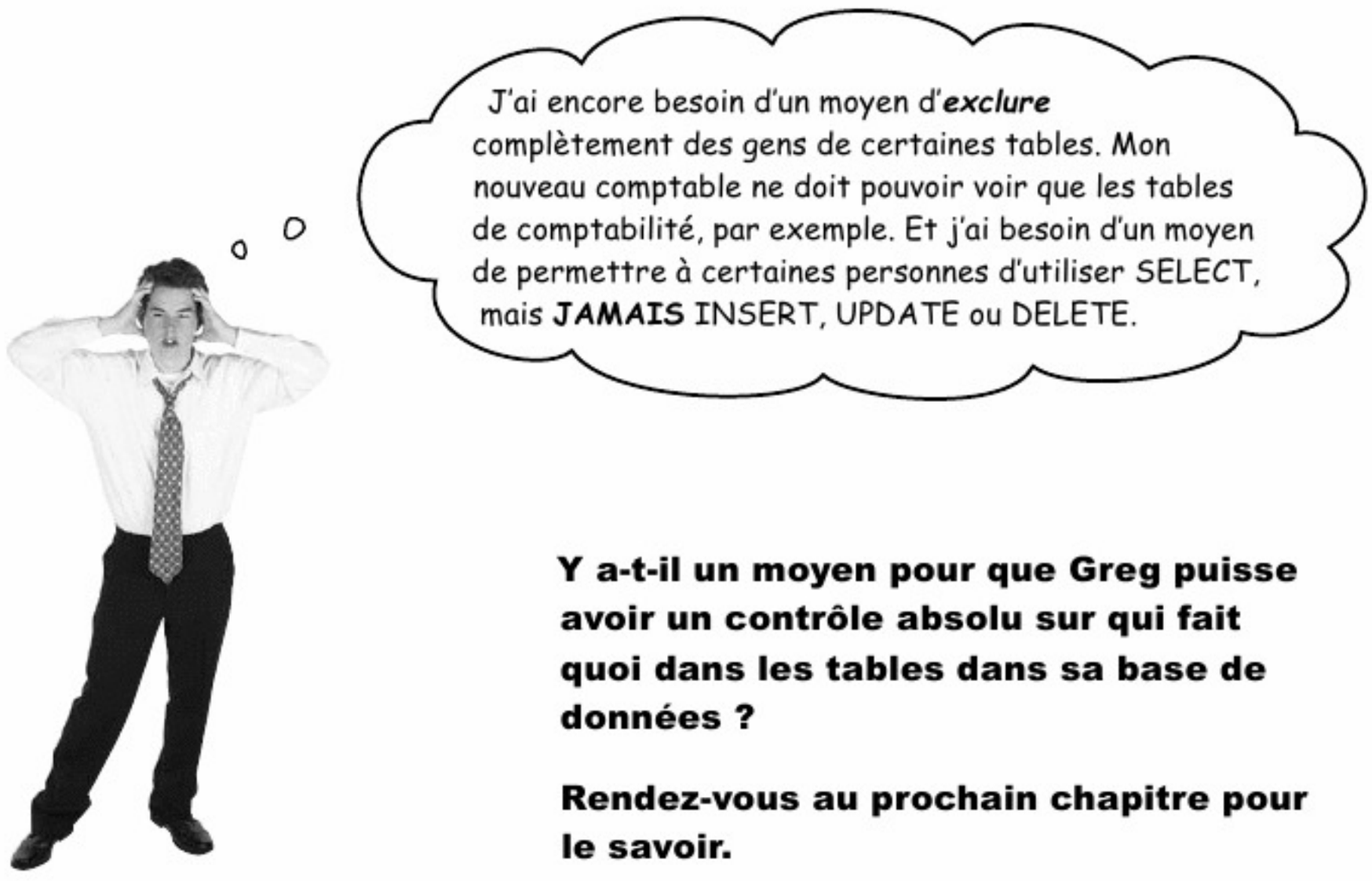
R: Vous devez dire à votre SGBDR que vous initiez une transaction avec `START TRANSACTION`. Il garde une trace du moment où la transaction a commencé pour savoir jusqu'où remonter pour tout annuler.

Q: Puis-je simplement utiliser START TRANSACTION pour essayer des requêtes ?

R: Vous pouvez et vous devriez le faire. C'est un excellent moyen de tester des requêtes qui modifient les données de vos tables sans modifier les tables de façon permanente si vous vous êtes trompé. Assurez-vous simplement de taper `COMMIT` ou `ROLLBACK` quand vous avez terminé.

Q: Pourquoi devrais-je me soucier de COMMIT ou ROLLBACK ?

R: Votre SGBDR garde une trace de tout ce qui a été fait quand vous êtes à l'intérieur d'une transaction. Cela s'appelle une session de transaction et plus vous en faites, plus elle s'agrandit. Il vaut mieux garder l'utilisation des transactions pour les fois où vous avez vraiment besoin de pouvoir annuler ce que vous faites. Cela évite de gaspiller de l'espace et de faire travailler votre SGBDR plus dur que nécessaire pour garder un suivi de ce que vous avez fait.



Y a-t-il un moyen pour que Greg puisse avoir un contrôle absolu sur qui fait quoi dans les tables dans sa base de données ?

Rendez-vous au prochain chapitre pour le savoir.

Boîte à outils SQL

Vous laissez le chapitre 11 derrière vous et votre boîte à outils est presque pleine. Vous avez appris à faire des VIEW de vos données et à exécuter des TRANSACTIONS. Pour une liste exhaustive des outils du livre, voyez l'annexe C.

VUES

Utilisez une vue pour traiter les résultats d'une requête comme une table. Génial pour transformer des requêtes complexes en requêtes simples.

VUES MODIFIABLES

Ce sont les vues qui vous permettent de modifier les données dans les tables sous-jacentes. Ces vues doivent contenir toutes les lignes NOT NULL de la ou des tables de base.

VUES NON-MODIFIABLES

Des vues qui ne peuvent pas être utilisées pour insérer ou supprimer des données dans la table de base.

TRANSACTIONS

C'est un groupe de requêtes qui doivent être exécutées ensemble comme une unité. Si elles ne peuvent pas toutes s'exécuter sans interruption, alors aucune ne l'est.

START TRANSACTION

Est utilisé pour dire au SGBDR de commencer une transaction. Rien n'est permanent jusqu'à ce que COMMIT soit envoyé. La transaction va continuer jusqu'à ce qu'une commande COMMIT ou ROLLBACK soit entrée, cette dernière renvoyant la base de données à son état antérieur à START TRANSACTION.

CONTRAINTES CHECK

Utilisez-les pour ne permettre qu'à certaines valeurs précises d'être insérées ou modifiées dans une table.

CHECK OPTION

Utilisez-le quand vous créez une vue modifiable pour forcer tous les insert et les update à remplir une clause WHERE dans la vue.

À vos crayons !

Solution

De la page 471.

S'il exécute le SELECT de la page 470 en utilisant la nouvelle vue augmentations, comment Norbert peut-il trier ses résultats avec leurs noms dans l'ordre alphabétique ?

En ajoutant un ORDER BY nom, soit à la vue quand elle est créée, soit au SELECT qui utilise la vue.

À vos crayons ! Solution

De la page 488.

Complétez le contenu de la tirelire après ces transactions. La voici comme elle est maintenant :

tirelire

id	piece	pays_piece
1	0.20	Portugal
2	1	Grèce
3	0.50	France
4	0.20	Belgique

```
START TRANSACTION;
UPDATE tirelire SET piece = 0.20 WHERE piece = 1
AND pays_piece < 'France';
COMMIT;
```

Pas de correspondance, donc pas de modification.

id	piece	pays_piece
1	0.20	Portugal
2	1	Grèce
3	0.50	France
4	0.20	Belgique

```
START TRANSACTION;
UPDATE tirelire SET piece = 0.50 WHERE piece = 0.20;
ROLLBACK;
```

Rollback, pas de modification.

id	piece	pays_piece
1	0.20	Portugal
2	1	Grèce
3	0.50	France
4	0.20	Belgique

```
START TRANSACTION;
UPDATE tirelire SET piece = 0.20 WHERE piece = 0.50
AND pays_piece < 'Portugal';
ROLLBACK;
```

Rollback, pas de modification.

id	piece	pays_piece
1	0.20	Portugal
2	1	Grèce
3	0.50	France
4	0.20	Belgique

```
START TRANSACTION;
UPDATE tirelire SET piece = 0.10 WHERE piece = 0.20
AND pays_piece > 'Espagne';
COMMIT;
```

Cette ligne est concernée.

id	piece	pays_piece
1	0.10	Portugal
2	1	Grèce
3	0.50	France
4	0.20	Belgique

```
START TRANSACTION;
UPDATE tirelire SET piece = 1 WHERE piece = 0.50
AND pays_piece < 'Grèce';
COMMIT;
```

Cette ligne est concernée.

id	piece	pays_piece
1	0.10	Portugal
2	1	Grèce
3	1	France
4	0.20	Belgique

Protéger vos biens

Vous avez investi beaucoup de temps et d'énergie pour créer votre base de données. Et vous seriez désolé si quelque chose lui arrivait. Vous avez aussi dû donner **accès à vos données** à d'autres personnes et vous avez peur qu'ils utilisent mal un insert ou un update, ou pire encore, un **delete**. Vous êtes sur le point d'apprendre comment les bases de données et les objets qui s'y trouvent peuvent devenir plus **sécurisés** et comment vous pouvez complètement contrôler ***qui fait quoi avec vos données.***

Problèmes d'utilisateur

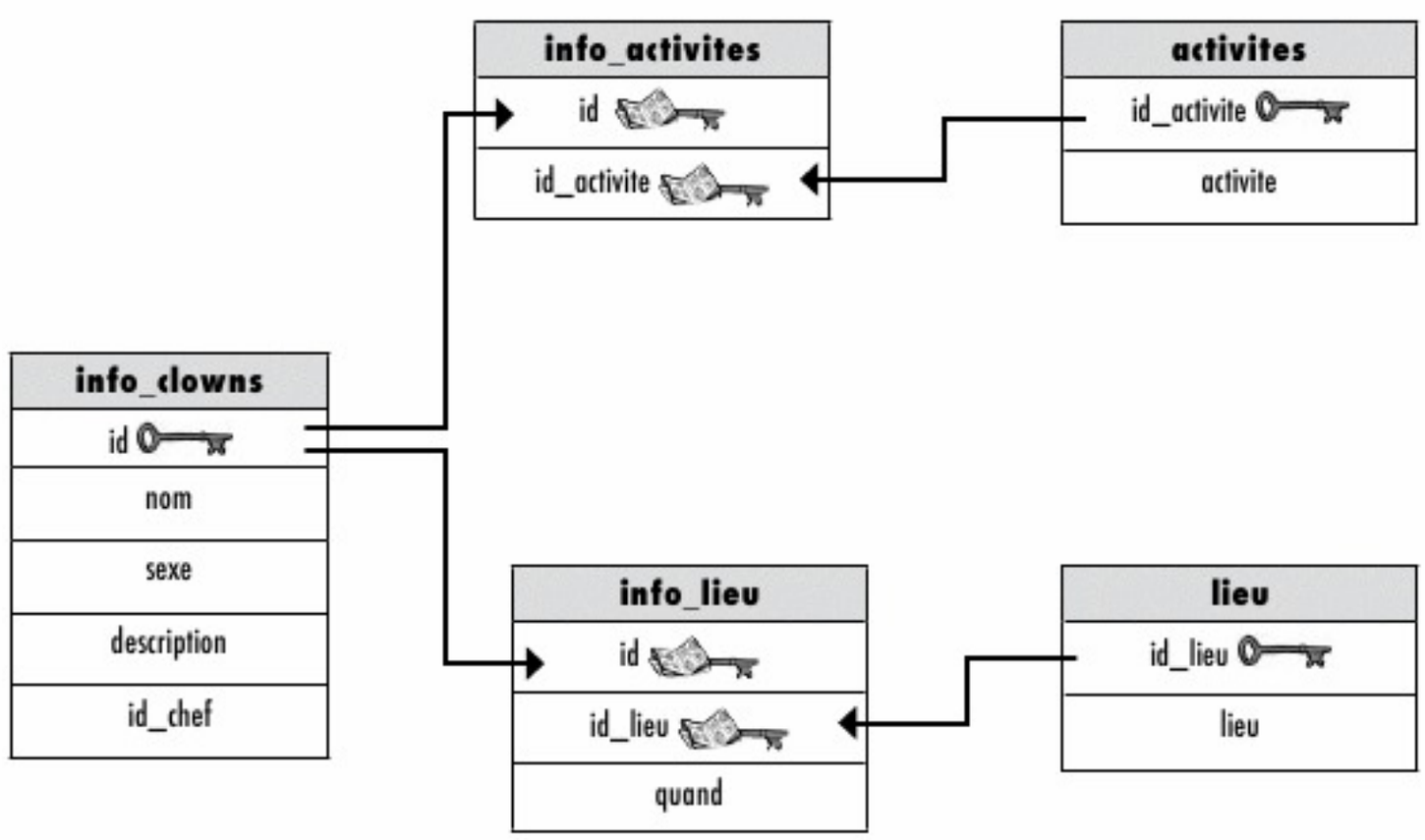
L'espionnage de clown est devenu un tel phénomène de masse que le conseil municipal de Dataville a dû engager toute une équipe pour pister les clowns et ajouter les données à la base de données `suivi_clowns`.

Malheureusement l'équipe a été infiltrée par un clown déguisé avec des vêtements normaux qui porte pour nom de code « Georges ». Il a provoqué de nombreux problèmes dans la base de données dont des pertes de données, des données modifiées et il a presque dupliqué des enregistrements, qui n'existent qu'à cause de ses erreurs délibérées dans leur orthographe. Voici quelques-uns des problèmes que présente la base de données de pistage des clowns :

Mme Câlin, Mme Coin et Me Colin ont toutes des lignes dans la table info_clowns. Nous sommes sûr qu'elles sont toutes le même clown parce que les colonnes sexe et description sont les mêmes (à part quelques mots déformés).

Avec ces entrées multiples dans la table info_clowns, les lieux actuels sont en désordre. La table info_lieu utilise les ID d'info_clowns pour Mme Câlin, Mme Coin et Me Colin.

La table activites est également pleine de mots déformés. Mme Câlin est une jongeleuse, Mme Coin est une jungleuze et Me Colin est une jugulaire.



Eviter les erreurs dans la base de données d'espionnage des clowns

Georges a démissionné avant que quiconque ait remarqué qu'il sabotait les données et maintenant nous ramassons les morceaux. À partir de maintenant, quand nous engageons de nouveaux employés, nous devons leur donner le pouvoir de `SELECTionner` dans la base de données pour qu'ils puissent identifier les clowns, mais nous voulons les empêcher d'`INSERER` des données. Ou d'utiliser `UPDATE`. Ou quoi que ce soit d'autre jusqu'à ce que nous ayons eu le temps de faire des vérifications approfondies sur leur identité.

Nous devons aussi faire attention : si l'on demande aux nouveaux employés d'utiliser `DELETE` sur nos données pour essayer de réparer les erreurs de Georges, ils peuvent supprimer de bonnes données en même temps que de mauvaises données.

Il est temps de protéger la base de données d'espionnage de clowns avant que d'autres clowns comme Georges ne la détruisent complètement.

À vos crayons !

Protégez la base de données d'espionnage des clowns contre de possibles sabotages. De chaque côté, écrivez des requêtes que les nouveaux employés doivent ou ne doivent pas être autorisés à faire. Incluez les noms des tables quand c'est possible.

Les nouveaux employés doivent pouvoir :

exemple : SELECT dans activites

Les nouveaux employés ne doivent **pas** pouvoir :

exemple : DROP TABLE sur info_clowns

À vos crayons ! Solution

Protégez la base de données d'espionnage des clowns contre de possibles sabotages. De chaque côté, écrivez des requêtes que les nouveaux employés doivent ou ne doivent pas être autorisés à faire. Incluez les noms des tables quand c'est possible.

Les nouveaux employés doivent pouvoir :

example : SELECT dans activites

SELECT dans info_clowns, info_activites, activites, info_lieu, lieu

Les nouveaux employés ne doivent **pas** pouvoir :

exemple : DROP TABLE sur info_clowns

DROP TABLE sur info_clowns, info_activites, activites, info_lieu, lieu

INSERT dans info_clowns, info_activites, activites, info_lieu, lieu

UPDATE dans info_clowns, info_activites, activites, info_lieu, lieu

ALTER TABLE sur info_clowns, info_activites, activites, info_lieu, lieu

DELETE dans info_clowns, info_activites, activites, info_lieu, lieu

Il y a de bonnes nouvelles, nous *pouvons* empêcher les clowns comme Georges de détruire nos données !

SQL nous donne la possibilité de contrôler ce que nos employés peuvent et ne peuvent pas faire dans la base de données d'espionnage de clowns. Mais avant de pouvoir faire cela, nous devons donner à Georges, ainsi qu'à tous les autres utilisateurs de notre base de données, un **compte utilisateur**.

Protéger le compte utilisateur racine

Jusqu'ici, nous n'avons eu qu'un utilisateur dans notre base de données et aucun mot de passe. Quiconque ayant accès au terminal ou à l'interface graphique de notre base de données a un contrôle absolu sur la base de données.

Par défaut, le premier utilisateur, l'utilisateur **root** (dit utilisateur racine ou super-utilisateur) a un contrôle absolu sur toute la base de données. C'est important, parce que l'utilisateur root doit pouvoir créer des comptes d'utilisateurs pour les autres utilisateurs. Nous ne voulons pas limiter ce que peut faire un utilisateur root, mais nous voulons certainement donner un mot de passe à notre compte root. Dans MySQL, la commande est simplement :

```
SET PASSWORD FOR 'root'@'localhost' = PASSWORD('b4dcl0wnZ');
```

Le nom d'utilisateur de notre utilisateur root est simplement 'root'.

'localhost' indique le lieu où le logiciel SQL est installé et exécuté.

C'est le mot de passe que nous avons choisi pour notre utilisateur root.

Les techniques d'autres SGBDR varient. Oracle utilise :

```
ALTER USER root IDENTIFIED BY nouveau-mot-de-passe;
```

Si vous utilisez une interface graphique vers votre base de données, vous trouverez probablement un moyen bien plus facile, à travers des dialogues, pour changer les mots de passe. ***L'important n'est pas tellement* comment *vous le faites, mais que vous* devez *absolument le faire.***

Consultez la documentation spécifique de votre SGBDR pour trouver les informations sur la protection du compte à la racine.

il n'y a pas de

Questions stupides

Q: Je ne comprends pas très bien ce que signifie « localhost ». Pouvez-vous expliquer cela plus en détail ?

R: `localhost` signifie que l'ordinateur que vous utilisez pour exécuter vos requêtes est le même ordinateur que celui sur lequel votre SGBDR SQL est installé. `localhost` est la valeur par défaut de ce paramètre, donc il est optionnel.

Q: Mais que se passe-t-il si j'utilise un client SQL sur une autre machine ?

R: C'est ce que l'on appelle un accès à distance. Vous devez dire à la requête où se trouve l'ordinateur. Vous pouvez le faire avec une adresse IP ou un nom d'hôte à la place de localhost. Par exemple, si votre logiciel SQL a été installé sur une machine appelée kumquats sur le réseau O'Reilly, vous pouvez utiliser quelque chose comme `root@kumquats.oreilly.com`. Mais celui-ci n'est pas un vrai serveur SQL, donc bien sûr cela ne fonctionnera pas.

Ajouter un nouvel utilisateur

Voici une question dont la réponse est évidente pour vous :

Comment SQL stocke-t-il les informations sur les utilisateurs, à votre avis ?

Dans une table, bien sûr ! SQL conserve une base de données sur lui-même. Elle comprend les id d'utilisateurs, les noms d'utilisateurs, les mots de passe et ce que chaque utilisateur est autorisé à faire dans chaque base de données.

Pour créer un nouvel utilisateur, nous pouvons commencer par un nom d'utilisateur et un mot de passe. Il n'y a pas de réelle commande SQL pour créer un utilisateur, mais la plupart des SGBDR utilisent quelque chose comme ça :

Voici le nom d'utilisateur de notre toute nouvelle employée, Elsie.

```
CREATE USER elsie
IDENTIFIED BY 'cl3v3rp4s5w0rd';
```

Voici son mot de passe.

SQL ne précise pas comment gérer les utilisateurs.

La création d'utilisateurs varie d'un SGBDR à l'autre. Vous devez consulter votre documentation pour trouver le bon moyen de créer un utilisateur dans votre SGBDR.

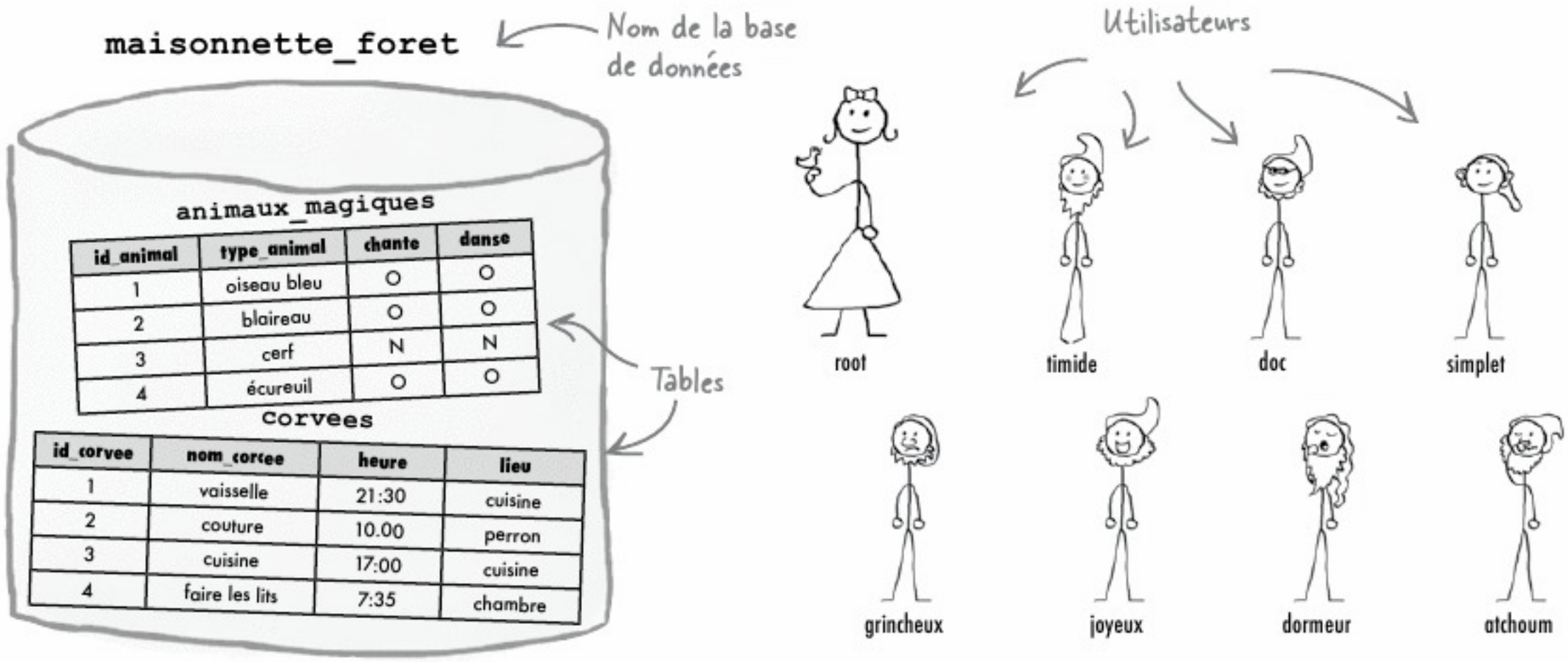
Nous aurions pu, mais nous ne savons pas toujours exactement quel accès nous devons accorder dès le début.

Nous devons encore décider exactement **à quoi notre utilisateur aura accès.** Nous allons faire une chose à la fois. Nous allons créer un utilisateur et ensuite lui accorder l'accès spécifique dont il a besoin. Après cela, nous assemblerons le tout avant de terminer. Savoir comment accorder un accès indépendamment de la création d'un utilisateur présente l'avantage de nous donner la possibilité de modifier les accès des utilisateurs plus tard, quand notre base de données change.

Décider exactement de quoi l'utilisateur a besoin

Nous avons créé le compte d'Elsie. Tel qu'il est pour l'instant, elle n'a aucune permission de faire quoi que ce soit. Nous devons utiliser une instruction **GRANT** pour ***lui donner une permission*** même pour un simple `SELECT` dans `info_clowns`.

Contrairement à notre compte root, qui a la permission d'exécuter n'importe quelle commande SQL sur n'importe quoi dans la base de données, les nouveaux utilisateurs que nous créons n'ont aucune permission. On peut utiliser l'instruction `GRANT` pour donner des droits spécifiques aux utilisateurs de nos bases de données. Voici ce que `GRANT` nous permet de faire :



Seuls certains utilisateurs peuvent modifier certaines tables.

Seule la personne responsable doit pouvoir ajouter de nouvelles corvées à la table corvees. Seul *root* peut utiliser `INSERT`, `UPDATE` et `DELETE` avec les corvées. Cependant, *joyeux* est responsable de la table `animaux_magiques` et peut `ALTERer` sa structure, ainsi qu'y exécuter n'importe quelle autre opération.

Les données d'une table précise ne doivent être accessibles qu'à certains utilisateurs.

Tout le monde sauf *grincheux* peut `SELECTionner` dans la table `animaux_magiques`. Il n'aime pas les animaux qui parlent.

Même *à l'intérieur des tables* on peut avoir besoin de permissions : certains utilisateurs peuvent voir certaines colonnes, mais pas d'autres.

Tout le monde sauf *simplet* peut voir la colonne instructions dans la table des corvées (cela le perturbe).

Vous pouvez contrôler précisément ce que peuvent faire les utilisateurs dans les tables et les colonnes avec l'instruction GRANT.

Une simple instruction GRANT

Nous savons qu'Elsie n'a, pour l'instant, la permission de rien faire. Elle peut ouvrir une session dans le logiciel SQL en utilisant son nom d'utilisateur et son mot de passe, mais c'est tout. Elle doit pouvoir `SELECTionner` dans la table `info_clowns`, donc nous pouvons lui donner cette **permission**. Nous devons accorder (`GRANT`) une permission à (`TO`) Elsie. Nous utilisons cette instruction :

On accorde à l'utilisateur la permission d'utiliser SELECT...

```
GRANT SELECT ON
info_clowns
TO elsie;
```

...sur la table nommée ici.

Et le nom d'utilisateur à qui nous accordons la permission est elsie.

Elsie a également besoin de la permission `SELECT` sur les autres tables d'espionnage des clowns pour pouvoir utiliser des jointures et des sous-requêtes dans ses instructions `SELECT`. Il nous faut une instruction `GRANT` pour chaque table :

```
GRANT SELECT ON activites TO elsie;
GRANT SELECT ON lieu TO elsie;
GRANT SELECT ON info_activites TO elsie;
GRANT SELECT ON info_lieu TO elsie;
```

Maintenant qu'Elsie est sous contrôle, essayez de trouver ce que font ces instructions GRANT dans la base de données maisonnette_foret que vous avez vue page 499.

	Le code	Que fait le code ?
1.	`GRANT INSERT ON animaux_ magiques TO doc;`	....
2.	`GRANT DELETE ON corvees TO joyeux, dormeur;`	....
3.	`GRANT DELETE ON corvees TO joyeux, dormeur WITH GRANT OPTION;`	....
4.	`GRANT SELECT(nom_corvee) ON corvees TO simplet;` Indice : c'est un nom de colonne.	....
5.	`GRANT SELECT, INSERT ON animaux_magiques TO atchoum;`	....
6.	`GRANT ALL ON animaux_ magiques TO timide;`	....

Maintenant essayez d'écrire vos propres instructions GRANT.

7.	....	Donne à doc la permission de `SELECT`ionner dans corvees.
8.	....	Donne à dormeur la permission d'utiliser `DELETE` dans `animaux_magiques` et lui donne aussi la permission d'accorder (`GRANT`) à quelqu'un d'autre le `DELETE` sur `animaux_magiques`.
9.	....	Donne toutes les permissions dans corvees à TOUS les utilisateurs.
10.	....	Ceci vous permet de définir le privilège `SELECT` de Doc en une fois pour toutes les tables de la base de données `maisonnette_foret`.

Solutions des exercices

Maintenant qu'Elsie est sous contrôle, essayez de trouver ce que font ces instructions GRANT dans la base de données maisonnette_foret que vous avez vue page 499.

Le code	Que fait le code ?
1. `GRANT INSERT ON animaux_magiques TO doc;`	Permet à doc d'utiliser INSERT dans la table animaux_magiques.
2. `GRANT DELETE ON corvees TO joyeux, dormeur;`	Permet à joyeux et dormeur d'utiliser DELETE dans la table corvees.
3. `GRANT DELETE ON corvees TO joyeux, dormeur WITH GRANT OPTION;`	Permet à joyeux et dormeur d'utiliser DELETE dans la table corvees et de donner à d'autres la même permission.
4. `GRANT SELECT(nom_corvee) ON corvees TO simplet;`	Permet à simplet d'utiliser SELECT dans la colonnes nom_corvee de la table corvees.
5. `GRANT SELECT, INSERT ON animaux_magiques TO atchoum;`	Permet à atchoum d'utiliser SELECT et INSERT dans la table animaux_magiques.
6. `GRANT ALL ON animaux_magiques TO timide;`	Permet à timide d'utiliser SELECT, UPDATE, INSERT et DELETE sur la table animaux_magiques.

Maintenant essayez d'écrire vos propres instructions GRANT.

7. GRANT SELECT ON corvees TO doc;	Donne à doc la permission de `SELECTionner` dans corvees.
8. GRANT DELETE ON animaux_magiques TO dormeur WITH GRANT OPTION;	Donne à dormeur la permission d'utiliser `DELETE` dans `animaux_magiques` et lui donne aussi la permission d'accorder (`GRANT`) à quelqu'un d'autre le `DELETE` sur `animaux_magiques`.
9. GRANT ALL ON corvees TO timide, doc, simplet, grincheux, joyeux, dormeur, atchoum;	Donne toutes les permissions dans corvees à TOUS les utilisateurs.
10. GRANT SELECT ON maisonnette_foret.* TO doc;	Ceci vous permet de définir le privilège `SELECT` de Doc en une fois pour toutes les tables de la base de données `maisonnette_foret`.

Variations sur GRANT

Dans l'exercice que vous venez de faire, vous avez vu les principales variations de l'instruction `GRANT`. Les voici :

1. **Vous pouvez nommer plusieurs utilisateurs dans la même instruction `GRANT`.**

   Chacun des utilisateurs nommés recevra la même permission.

2. **`WITH GRANT OPTION` donne la permission aux utilisateurs de donner à d'autres utilisateurs la permission qu'ils viennent de recevoir.**

   Cela peut sembler déroutant, mais cela signifie simplement que si l'utilisateur a reçu une permission `SELECT` dans `corvees`, il peut donner à n'importe quel autre utilisateur cette même permission d'utiliser `SELECT` dans `corvees`.

3. **On peut utiliser une ou plusieurs colonnes précises dans une table au lieu de la table toute entière.**

   On peut donner la permission de n'utiliser `SELECT` que dans une colonne. Les seuls résultats que l'utilisateur pourra voir seront ceux de cette colonne.

4. **On peut spécifier plus d'une permission sur une table.**

   Faites simplement la liste des permissions que vous voulez accorder sur une table en utilisant une virgule après chacune d'entre elles.

5. **`GRANT ALL` donne aux utilisateurs la permission d'utiliser `SELECT`, `UPDATE`, `INSERT` et `DELETE` dans la table indiquée.**

   C'est simplement un raccourci pour dire « donne aux utilisateurs la permission d'utiliser `SELECT`, `UPDATE`, `INSERT` et `DELETE` dans la table indiquée ».

6. **On peut indiquer toutes les tables d'une base de données avec `nom_bdd.*`**

   Tout comme vous utilisez le joker * dans une instruction SELECT, ceci désigne toutes les tables d'une base de données.

REVOKE révoque les privilèges

Supposez que nous décidions d'enlever le privilège `SELECT` que nous avons donné à Elsie. Pour ce faire, nous avons besoin de l'instruction **REVOKE**.

Vous rappelez-vous de notre simple instruction `GRANT` ? La syntaxe de `REVOKE` est presque identique. Au lieu du mot « *grant* », on a « *revoke* » et au lieu de « *to* », nous utilisons « *from* ».

Nous enlevons le privilège SELECT.

```
REVOKE SELECT ON
info_clowns
FROM elsie;
```

Nous utilisons revoke from un utilisateur au lieu de grant to.

Vous pouvons également ne révoquer que `WITH GRANT OPTION` **mais garder le privilège intact.** Dans cet exemple, *joyeux* et *dormeur* peuvent toujours utiliser `DELETE` dans la table corvees, mais ils ne peuvent plus donner ce privilège à quelqu'un d'autre :

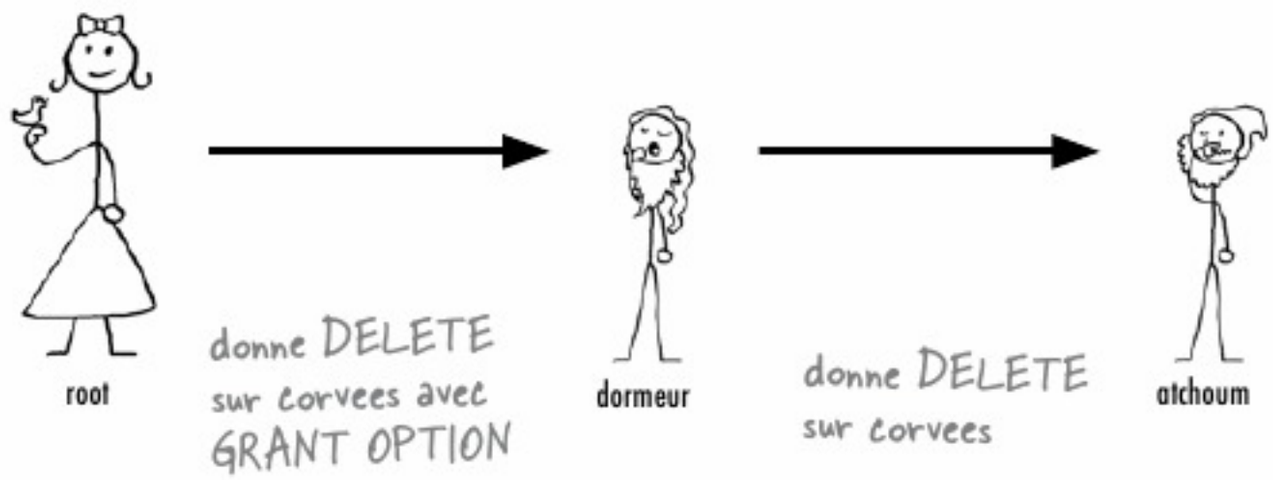
Nous enlevons seulement le privilège GRANT OPTION.

```
REVOKE GRANT OPTION ON
DELETE ON corvees
FROM joyeux, dormeur;
```

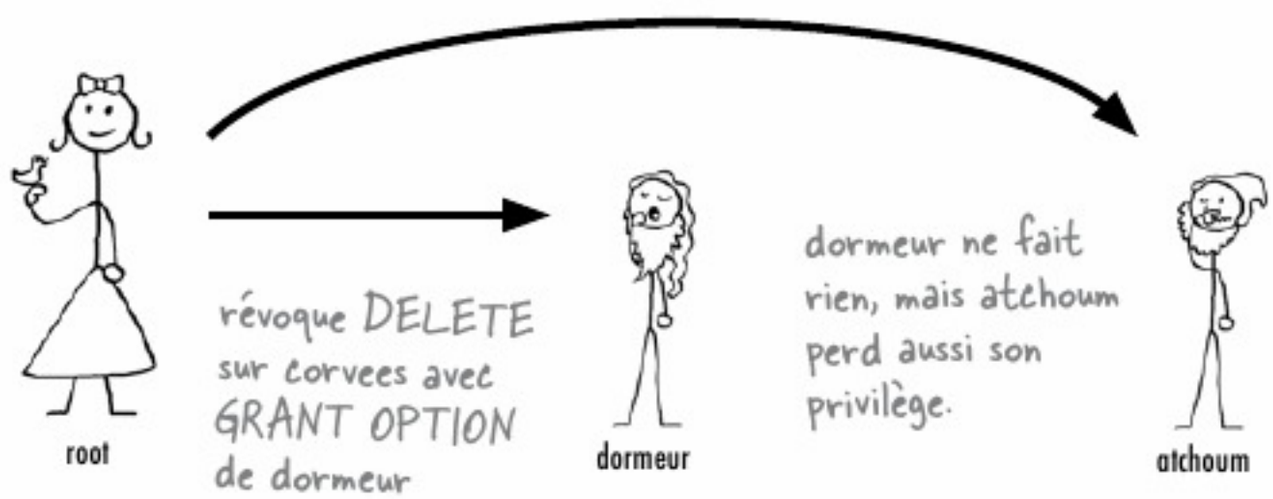
Les utilisateurs joyeux et dormeur peuvent toujours utiliser DELETE, mais ils ne peuvent plus accorder ce privilège à quelqu'un d'autre.

RÉVOQUER une GRANT OPTION déjà utilisée

Prenez ce scénario. L'utilisatrice root a donné à *dormeur* le privilège `DELETE` avec `GRANT OPTION` sur la table corvees. Puis *dormeur* a donné les privilèges `DELETE` à *atchoum*, sur corvees également.



Supposez que l'utilisatrice root change d'avis et reprenne le privilège à *dormeur*. Il sera aussi enlevé à *atchoum*, **même si elle ne l'a retiré qu'à *dormeur***.



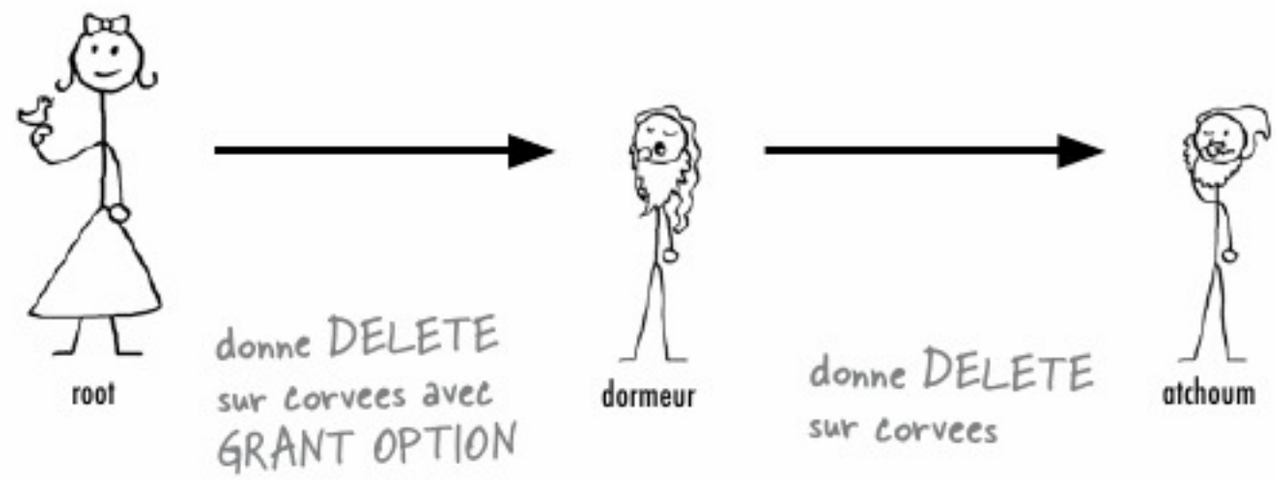
Un effet secondaire de l'instruction `REVOKE` est qu'*atchoum* perd aussi le privilège. Il existe deux mots-clés que l'on peut utiliser pour contrôler ce qui se passe quand on révoque.

MUSCLEZ VOS NEURONES

Vous êtes sur le point de découvrir les mots-clés `RESTRICT` et `CASCADE`. Que pensez-vous qu'ils fassent ?

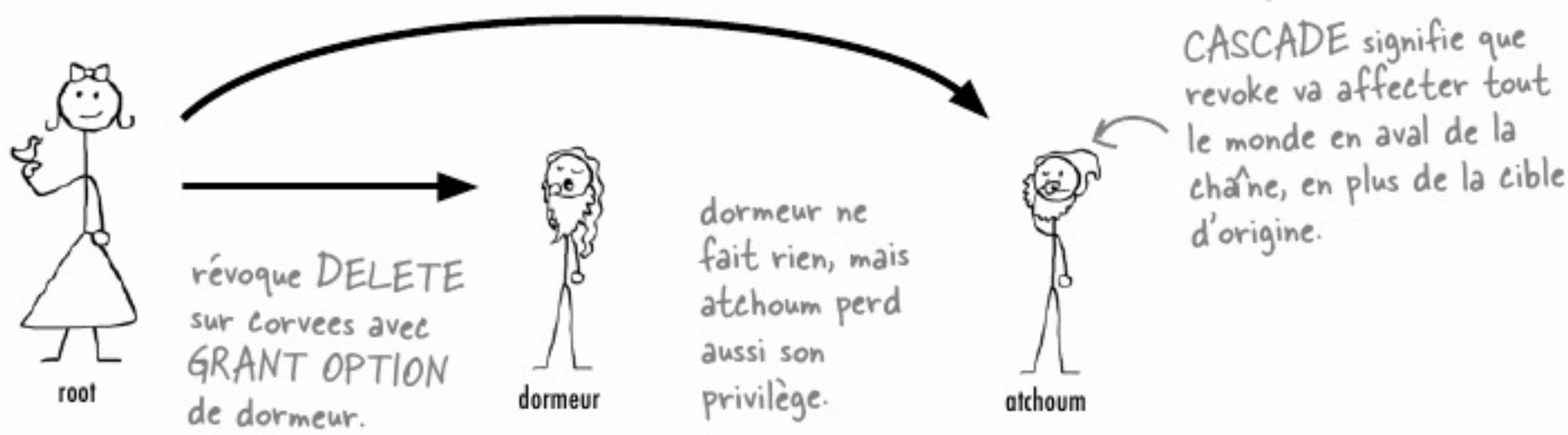
REVOKE avec précision

Il y a deux moyens de révoquer des privilèges en s'assurant que vous n'affectez pas d'autres utilisateurs que celui que vous voulez affecter. Vous pouvez utiliser les mots-clés CASCADE et RESTRICT pour cibler plus précisément les utilisateurs qui gardent ou qui perdent leurs privilèges.



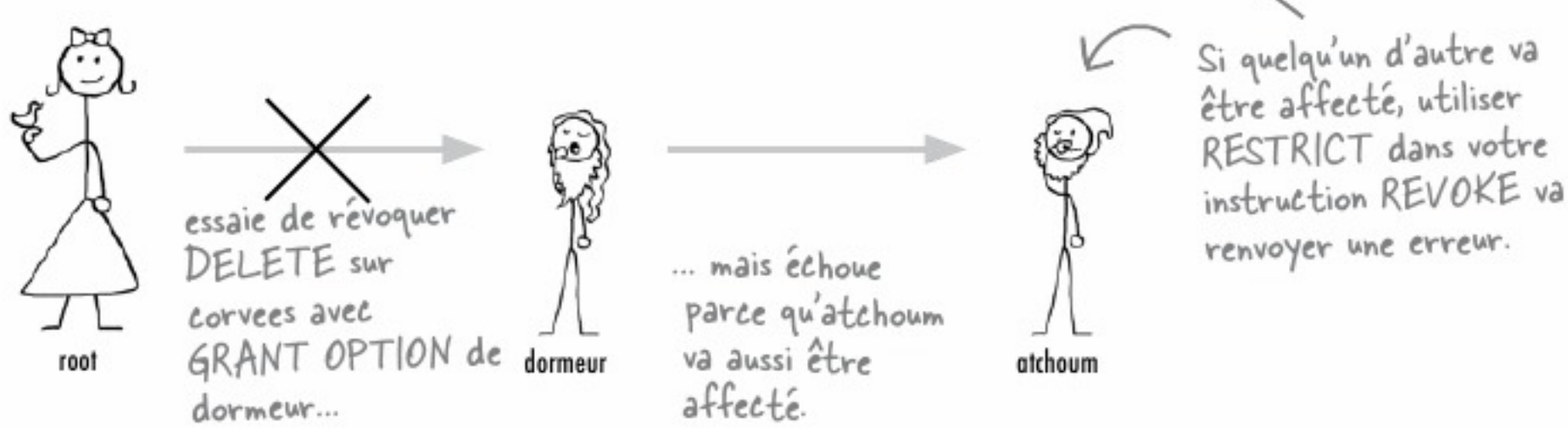
Le premier, CASCADE, retire le privilège à l'utilisateur que vous ciblez (ici, dormeur) ainsi qu'à quiconque à qui cet utilisateur a accordé cette permission.

```
REVOKE DELETE ON corvees FROM dormeur CASCADE;
```



Utiliser RESTRICT quand vous voulez retirer un privilège à un utilisateur va renvoyer une erreur si cet utilisateur a accordé des privilèges à quelqu'un d'autre.

```
REVOKE DELETE ON corvees FROM dormeur RESTRICT;
```



Tous deux gardent leurs privilèges et *root* reçoit une erreur. Elle est empêchée de faire la modification et reçoit une erreur parce que cela va aussi avoir un effet sur *atchoum*.

À vos crayons !

Quelqu'un n'arrête pas de donner les mauvais privilèges à Elsie. Écrivez les instructions REVOKE appropriées pour la renvoyer à un statut plus prudent de simple SELECT.

```
GRANT SELECT, INSERT, DELETE ON lieu TO elsie;
```

. .....

```
GRANT ALL ON info_clowns TO elsie;
```

. .....

```
GRANT SELECT, INSERT ON activites TO elsie;
```

. .....

```
GRANT DELETE, SELECT on info_lieu TO elsie WITH
GRANT OPTION;
```

. .....

```
GRANT INSERT(lieu), DELETE ON lieu TO elsie;
```

. .....

À vos crayons ! Solution

Quelqu'un n'arrête pas de donner les mauvais privilèges à Elsie. Écrivez les instructions REVOKE appropriées pour la renvoyer à un statut plus prudent de simple SELECT.

```
GRANT SELECT, INSERT, DELETE ON lieu TO elsie;
```

REVOKE INSERT, DELETE ON lieu FROM elsie;

```
GRANT ALL ON info_clowns TO elsie;
```

REVOKE INSERT, UPDATE, DELETE ON info_clowns FROM elsie;

Nous voulons lui laisser ses privilèges SELECT, donc nous ne révoquons pas tout.

```
GRANT SELECT, INSERT ON activites TO elsie;
```

REVOKE INSERT ON activites FROM elsie;

On aurait aussi pu faire cela en révoquant tout puis en accordant ce que l'on veut avec GRANT.

```
GRANT DELETE, SELECT on info_lieu TO elsie WITH
GRANT OPTION;
```

REVOKE DELETE on info_lieu FROM elsie CASCADE;

Nous ferions mieux de nous assurer qu'elle n'a pas donné à quelqu'un d'autre les privilèges qu'elle avait.

```
GRANT INSERT(lieu), DELETE ON lieu TO elsie;
```

REVOKE INSERT(lieu), DELETE ON lieu FROM elsie;

Il semble qu'on pourrait aussi utiliser GRANT ici pour vérifier qu'elle peut toujours SELECTionner dans lieu.

il n'y a pas de Questions stupides

Q: Je repense aux instructions GRANT qui précisent des noms de colonnes. Qu'arrive-t-il si on accorde INSERT sur une seule colonne d'une table ?

R: Bonne question. C'est en fait un privilège plutôt inutile. Si vous ne pouvez mettre une valeur que dans une seule colonne, vous ne pouvez pas insérer de vraies lignes dans la table. La seule façon pour que cela fonctionne est que la table n'ait que la colonne précisée dans l'instruction GRANT.

Q: Existe-t-il d'autres instructions GRANT tout aussi inutiles ?

R: Presque tous les privilèges par colonne sont assez inutiles à moins qu'ils ne soient utilisés avec un SELECT dans l'instruction GRANT.

Q: Supposez que je veuille ajouter un utilisateur et le laisser SELECTionner dans toutes les tables de mes bases de données. Y a-t-il un moyen facile de le faire ?

R: Comme beaucoup de choses dans ce chapitre, cela dépend de votre SGBDR. Vous pouvez accorder des privilèges globaux dans MySQL comme ceci :

```
GRANT SELECT ON *.*
TO elsie;
```

La première astérisque fait référence à toutes les bases de données, la seconde à toutes les tables.

Q: Est-ce que CASCADE est le paramètre par défaut si l'on ne précise pas comment on veut révoquer ?

R: Généralement CASCADE est l'option par défaut, mais encore une fois, consultez la documentation de votre SGBDR pour les précisions.

Q: Que se passe-t-il si je REVOKE quelque chose que l'utilisateur n'a pas ?

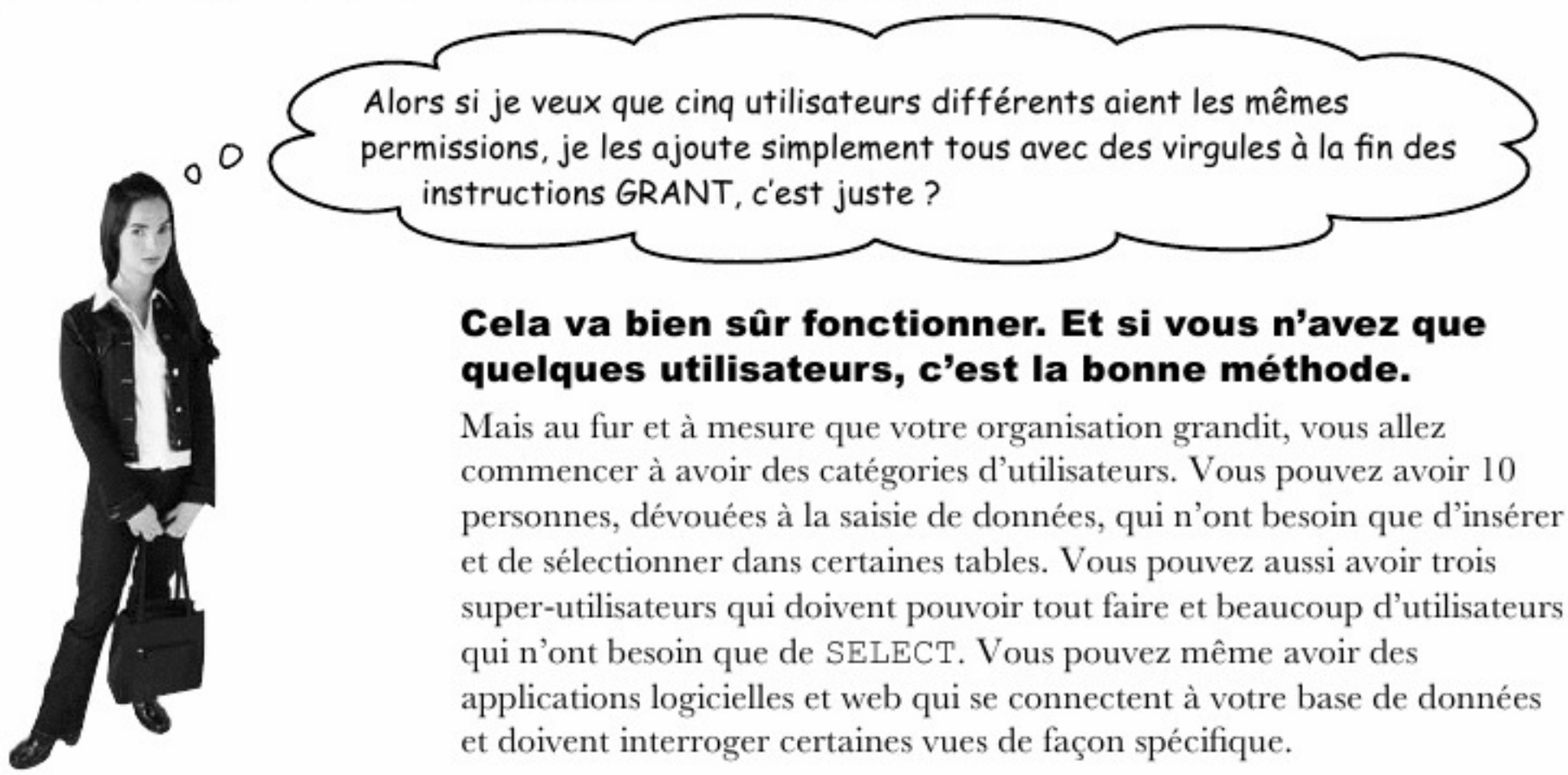
R: Vous aurez simplement une erreur vous disant que le GRANT n'existe pas.

Q: Que se passe-t-il si deux personnes différentes donnent à l'utilisateur *atchoum* le même privilège que celui que révoque *root* dans l'exemple précédent ?

R: C'est là que les choses se compliquent. Certains systèmes ne feront pas attention à l'origine du GRANT quand on utilise CASCADE et certains vont ignorer CASCADE. Mais c'est là encore l'occasion de consulter la documentation de votre logiciel.

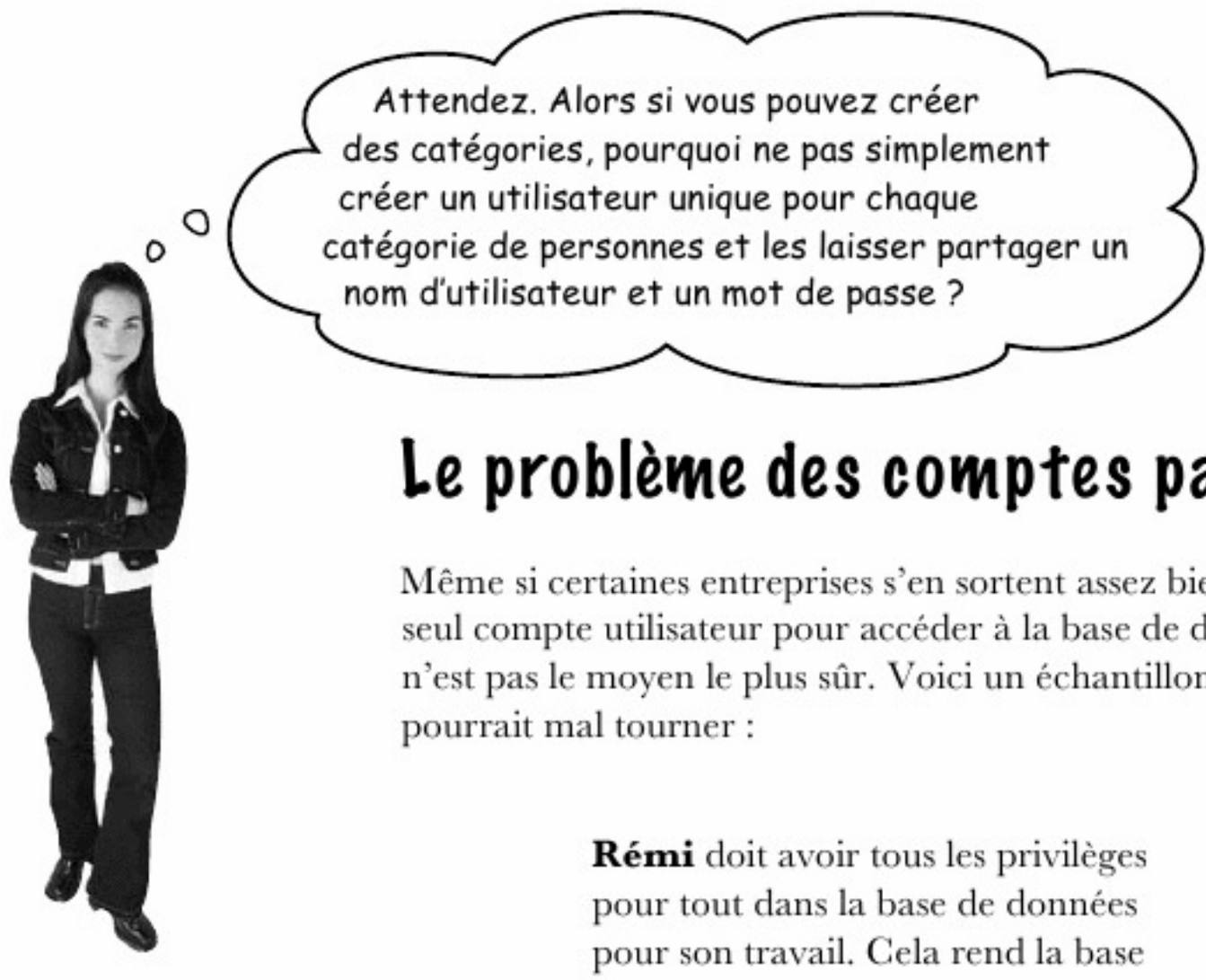
Q: Y a-t-il autre chose, en plus des tables et des colonnes, avec quoi je peux utiliser GRANT et REVOKE ?

R: Vous pouvez les utiliser avec les vues, de la même façon qu'avec les tables, à moins que la vue soit non modifiable. Dans ce cas, vous ne pourriez pas utiliser INSERT, même si vous en aviez la permission. Et tout comme dans une table, vous pouvez accorder l'accès à des colonnes précises dans une vue.



Cela va bien sûr fonctionner. Et si vous n'avez que quelques utilisateurs, c'est la bonne méthode.

Mais au fur et à mesure que votre organisation grandit, vous allez commencer à avoir des catégories d'utilisateurs. Vous pouvez avoir 10 personnes, dévouées à la saisie de données, qui n'ont besoin que d'insérer et de sélectionner dans certaines tables. Vous pouvez aussi avoir trois super-utilisateurs qui doivent pouvoir tout faire et beaucoup d'utilisateurs qui n'ont besoin que de SELECT. Vous pouvez même avoir des applications logicielles et web qui se connectent à votre base de données et doivent interroger certaines vues de façon spécifique.



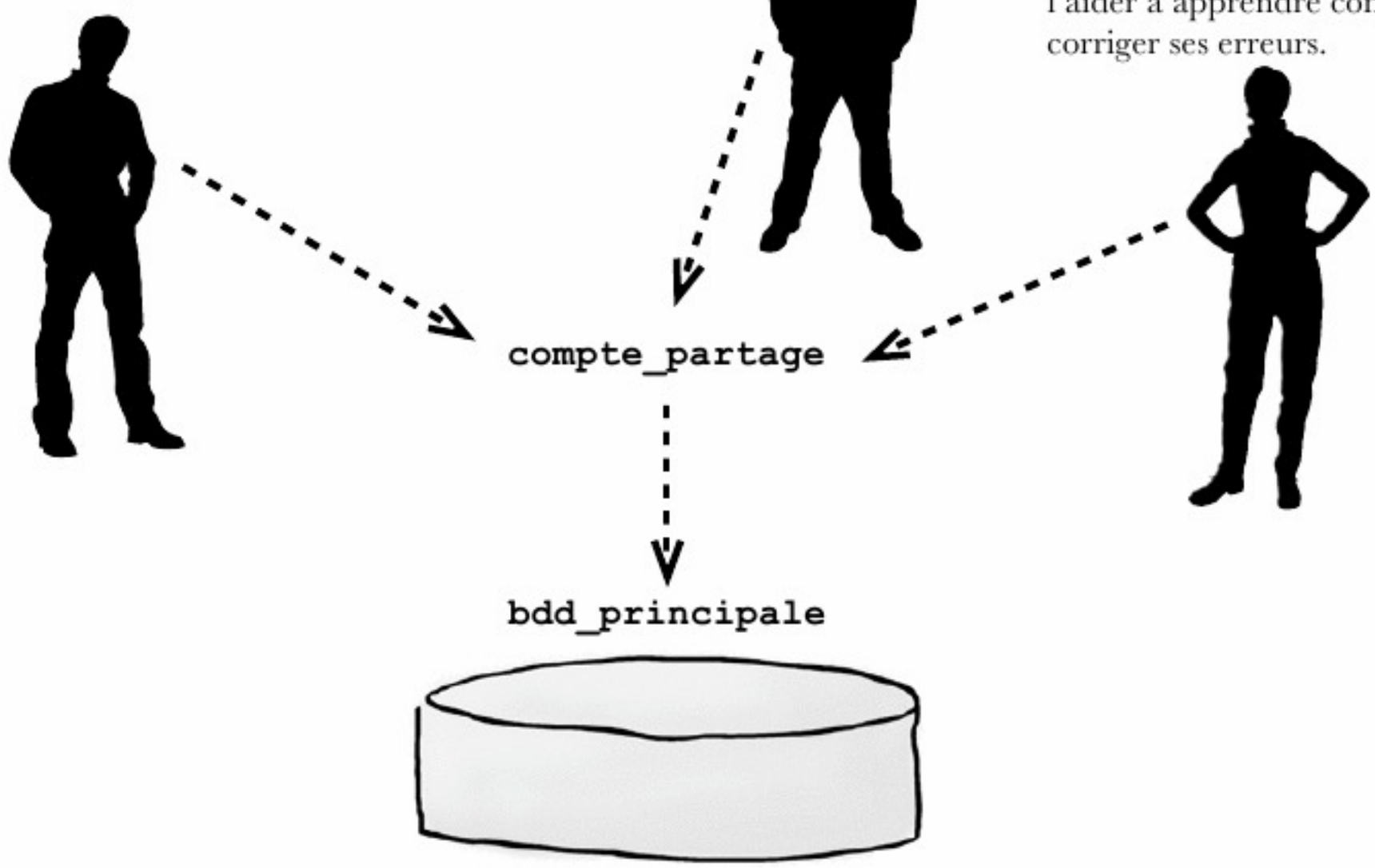
Le problème des comptes partagés

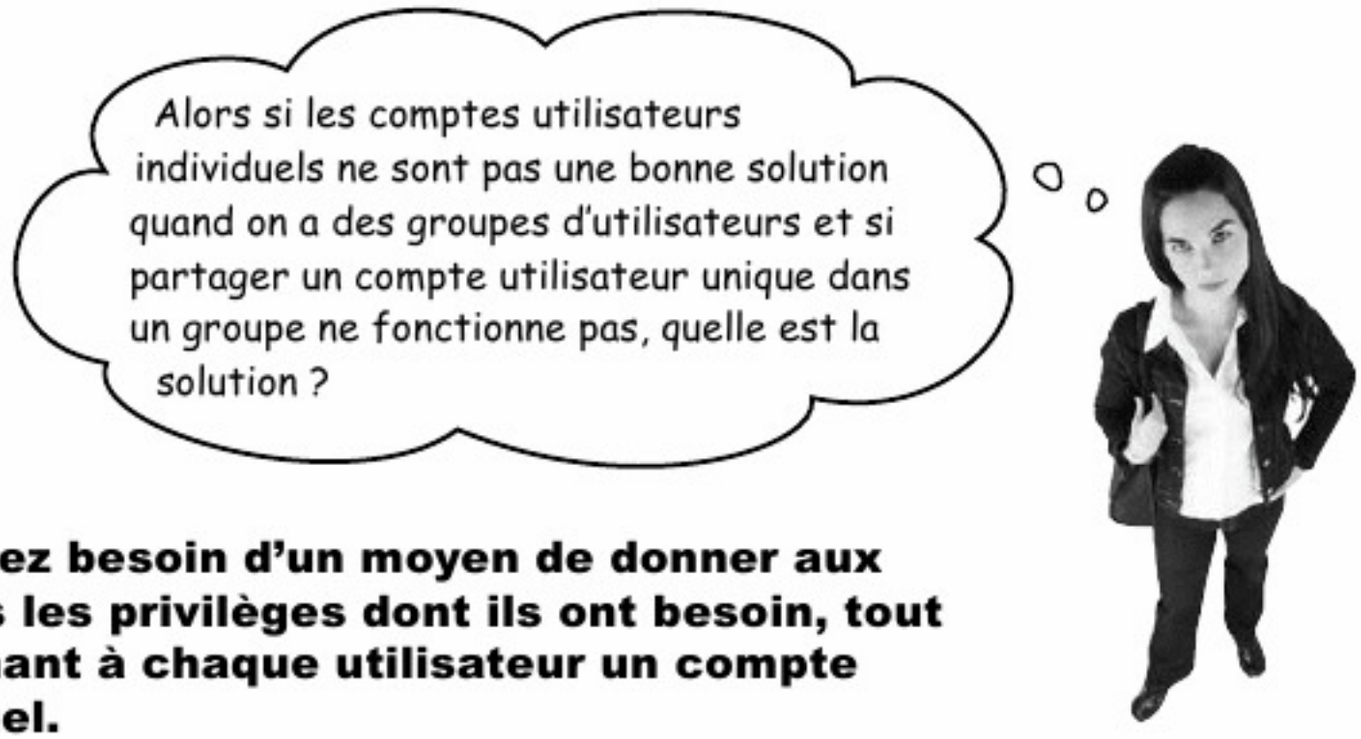
Même si certaines entreprises s'en sortent assez bien avec un seul compte utilisateur pour accéder à la base de données, ce n'est pas le moyen le plus sûr. Voici un échantillon de ce qui pourrait mal tourner :

Rémi doit avoir tous les privilèges pour tout dans la base de données pour son travail. Cela rend la base de données vulnérables aux autres utilisateurs qui ne sont pas aussi versés que lui en SQL et plus enclins à faire des erreurs.

Simon change le mot de passe et oublie de le dire aux autres. Personne ne peut accéder à la base de données jusqu'à ce qu'il se rappelle de les prévenir.

Paula ne maîtrise pas bien l'écriture des instructions update et n'arrête pas de semer la pagaille dans les données. Personne ne sait qui sème la pagaille dans les données, donc personne ne peut l'aider à apprendre comment corriger ses erreurs.





Vous avez besoin d'un moyen de donner aux groupes les privilèges dont ils ont besoin, tout en donnant à chaque utilisateur un compte individuel.

Vous avez besoin des **rôles**. Un rôle est un moyen de regrouper des privilèges spécifiques et de les appliquer à tous les membres d'un groupe. Votre rôle devient un objet de votre base de données que vous pouvez modifier si nécessaire quand votre base de données change, sans devoir modifier explicitement les privilèges de tous les utilisateurs pour refléter les changements de la base de données.

Et définir un rôle est très simple :

```
CREATE ROLE saisie_donnees;
```

Le nom du rôle que nous créons

Il n'y a pas de rôles dans MySQL.

Les rôles sont une fonctionnalité qui sera sans doute intégrée à une future version de MySQL, mais pour l'instant, vous devrez assigner vos privilèges pour chaque utilisateur.

Pour ajouter des privilèges au rôle, vous le traitez simplement comme un nom d'utilisateur :

```
GRANT SELECT, INSERT ON une_table TO saisie_donnees;
```

Au lieu d'un utilisateur, nous utilisons le nom du rôle quand nous assignons des privilèges.

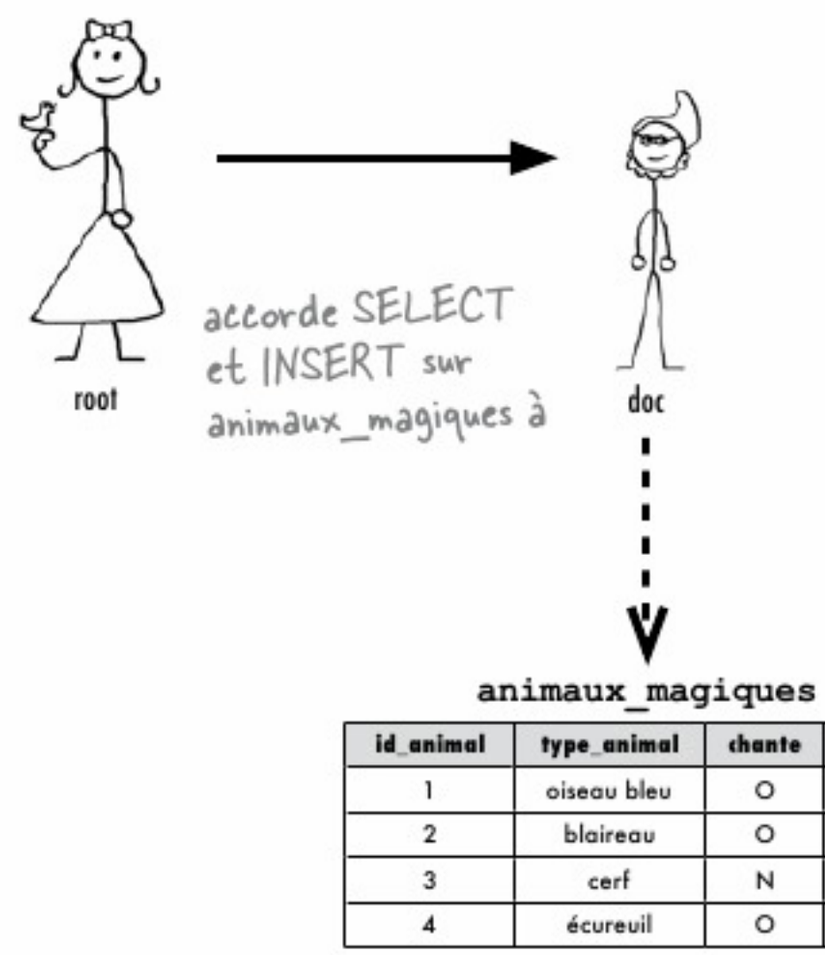
Nous avons créé notre rôle et lui avons donné des privilèges. Maintenant nous devons l'assigner à quelqu'un...

Utiliser votre rôle

Avant de créer notre rôle, nous aurions pu donner des privilèges à nos utilisateurs de saisie de données directement en utilisant des instructions GRANT, comme :

```
GRANT SELECT, INSERT
ON animaux_magiques
TO doc;
```

L'ancienne méthode.

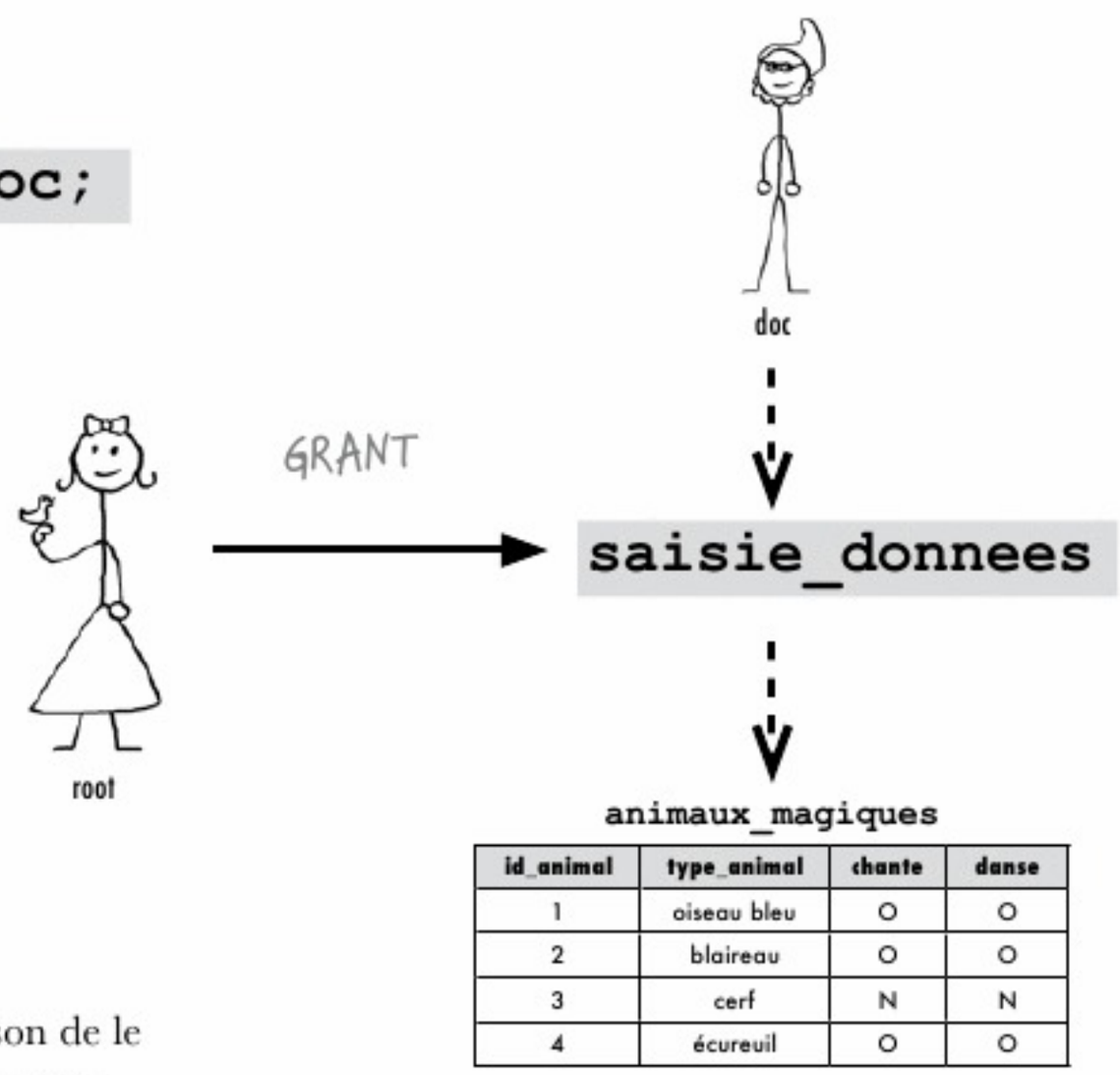


id_animal	type_animal	chante	danse
1	oiseau bleu	O	O
2	blaireau	O	O
3	cerf	N	N
4	écureuil	O	O

Maintenant nous n'avons qu'à substituer notre nouveau rôle à l'opération GRANT et l'appliquer à *doc*. Nous n'avons pas besoin de mentionner les privilèges ou la table parce que tout est stocké dans le rôle saisie_données :

```
GRANT saisie_donnees TO doc;
```

Le nom du rôle remplace le nom de la table et les privilèges.



id_animal	type_animal	chante	danse
1	oiseau bleu	O	O
2	blaireau	O	O
3	cerf	N	N
4	écureuil	O	O

Suppression de rôles

Si vous n'avez plus besoin de votre rôle, il n'y a aucune raison de le garder. Utilisez une instruction DROP pour vous en débarrasser :

```
DROP ROLE saisie_donnees;
```

il n'y a pas de Questions stupides

Q: Si je veux donner des privilèges avec GRANT pour toutes les tables d'une base de données ? Dois-je toutes les taper ?

R: Non, vous pouvez utiliser cette syntaxe :

```
GRANT SELECT, INSERT, DELETE
ON liste_greg.*
TO martin;
```

Nommez simplement la base de données et utilisez * pour assigner les privilèges à toutes les tables de cette base de données.

Q: Si un rôle est assigné à un utilisateur, peut-on le supprimer quand même ?

R: Vous pouvez supprimer des rôles qui sont utilisés. Faites très attention en supprimant un rôle de ne pas enlever aux utilisateurs les permissions dont ils ont besoin.

Q: Cela signifie que si un utilisateur a un rôle qui est supprimé, il perd ces permissions ?

R: C'est tout à fait exact. C'est comme si vous lui aviez explicitement accordé puis révoqué ces permissions. Seulement au lieu d'affecter un seul utilisateur quand vous utilisez REVOKE FROM quelqu'un, vous aurez des conséquences sur les permissions de tous les utilisateurs ayant reçu ce rôle.

Q: Un utilisateur peut-il avoir plus d'un rôle à la fois ?

R: Oui. Assurez-vous seulement qu'ils n'ont pas de permissions conflictuelles ou vous pourriez vous créer des problèmes. Les permissions refusées ont la priorité sur celles qui sont accordées.

Révoquer votre rôle

Révoquer un rôle fonctionne comme révoquer une permission. Voyez si vous pouvez écrire l'instruction pour révoquer saisie_données de Doc ***sans revenir en arrière dans le chapitre***.

À vos crayons ! Solution

Révoquer un rôle fonctionne comme révoquer une permission. Voyez si vous pouvez écrire l'instruction pour révoquer saisie_données de Doc ***sans revenir en arrière dans le chapitre***.

```
REVOKE saisie_donnees FROM doc;
```

Utiliser votre rôle WITH ADMIN OPTION

Tout comme l'instruction `GRANT` a une `WITH GRANT OPTION`, un rôle a la similaire `WITH ADMIN OPTION`. Cette option permet à quiconque avec ce rôle de l'accorder à quelqu'un d'autre. Par exemple, si nous utilisons cette instruction :

```
GRANT saisie_donnees TO doc WITH ADMIN OPTION;
```

WITH ADMIN OPTION permet à l'utilisateur doc d'accorder le rôle saisie_donnees à quelqu'un d'autre.

doc a maintenant des privilèges administrateur et il peut accorder à *joyeux* le rôle `saisie_donnees` de la même façon qu'il le lui a été :

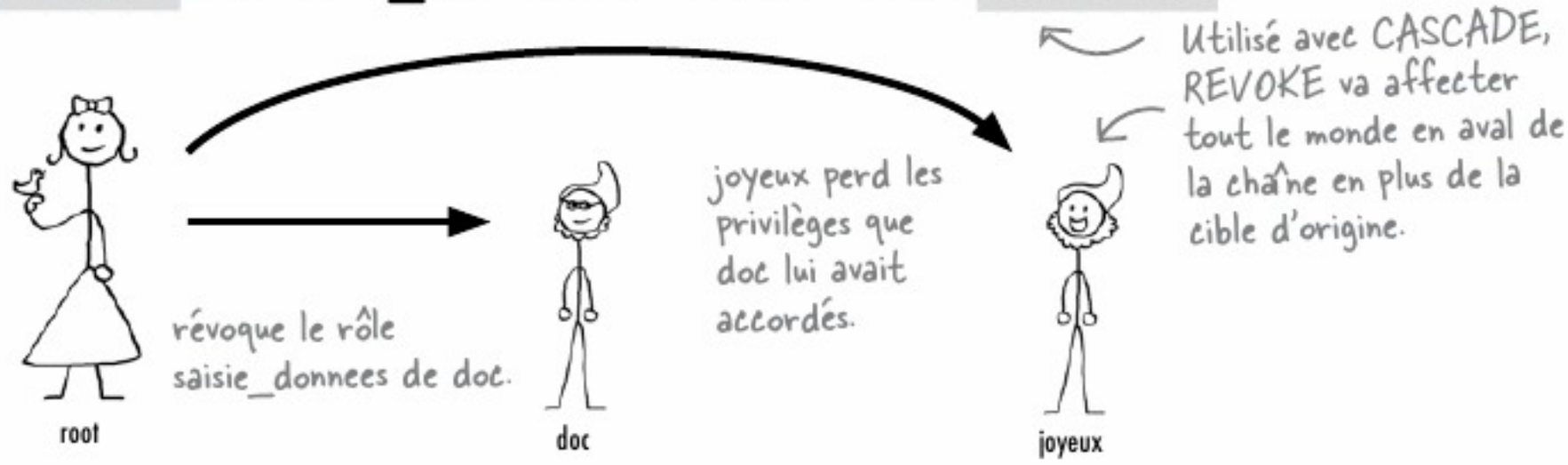
```
GRANT saisie_donnees TO joyeux;
```

Quand elle est utilisée avec un rôle, la commande `REVOKE` a les mêmes mots-clés `CASCADE` et `RESTRICT`. Regardons comment ils fonctionnent :

RÉVOQUER un rôle avec CASCADE

Utilisé avec `CASCADE`, `REVOKE` affecte tout le monde en aval de la chaîne en plus de la cible d'origine :

```
REVOKE saisie_donnees FROM doc CASCADE;
```

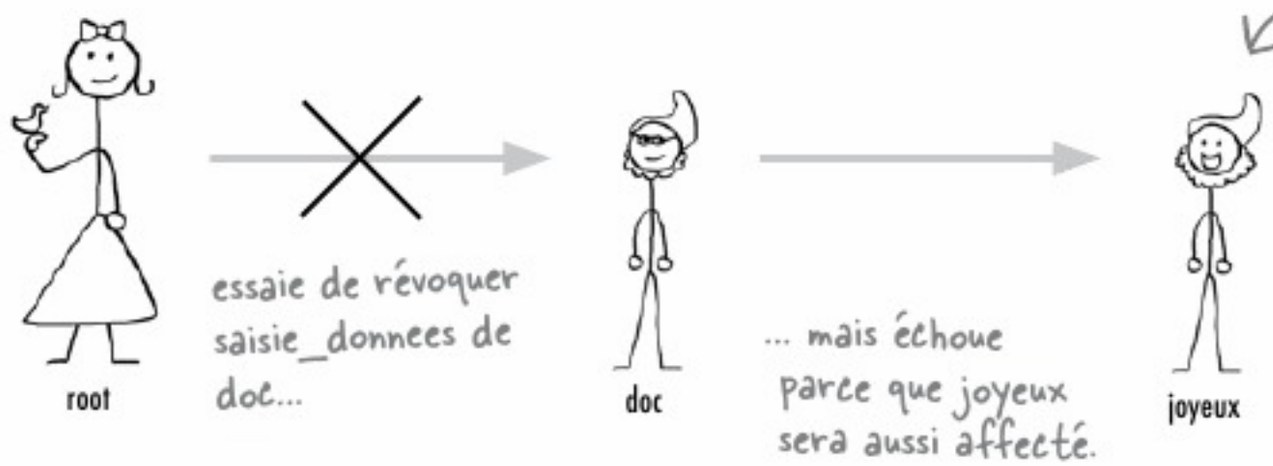


RÉVOQUER un rôle avec RESTRICT

Utiliser `RESTRICT` quand vous voulez retirer un privilège à un utilisateur va renvoyer une erreur si cet utilisateur a accordé des privilèges à quelqu'un d'autre.

```
REVOKE saisie_donnees FROM doc RESTRICT;
```

Si quelqu'un d'autre va être affecté, utiliser RESTRICT dans votre instruction REVOKE va renvoyer une erreur.



Tous deux conservent leurs privilèges et *root* reçoit une erreur. Elle est empêchée de faire la modification parce que cela va aussi avoir un effet sur *joyeux*.

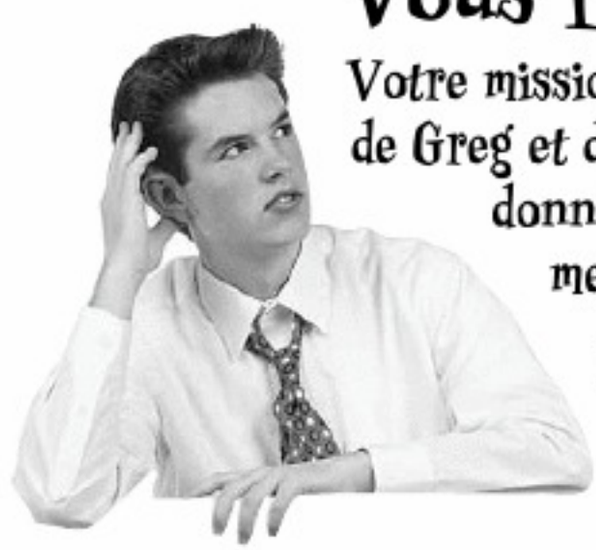
Les rôles ont l'air supers, mais pouvons-nous revenir à la réalité une minute ? Je n'ai que deux employés, bientôt trois. Je ne veux pas de rôles, mais je veux qu'ils arrêtent d'utiliser le compte root. Je réalise mon erreur. Pouvez-vous m'aider à leur accorder l'accès correct sans utiliser de rôles ?

Oui, il est temps de sécuriser la liste_greg en définissant les permissions des employés de Greg.

Greg va avoir besoin de parcourir les étapes de ce chapitre pour protéger le compte utilisateur, trouver ce dont ont besoin ses employés et leur donner les bons privilèges.

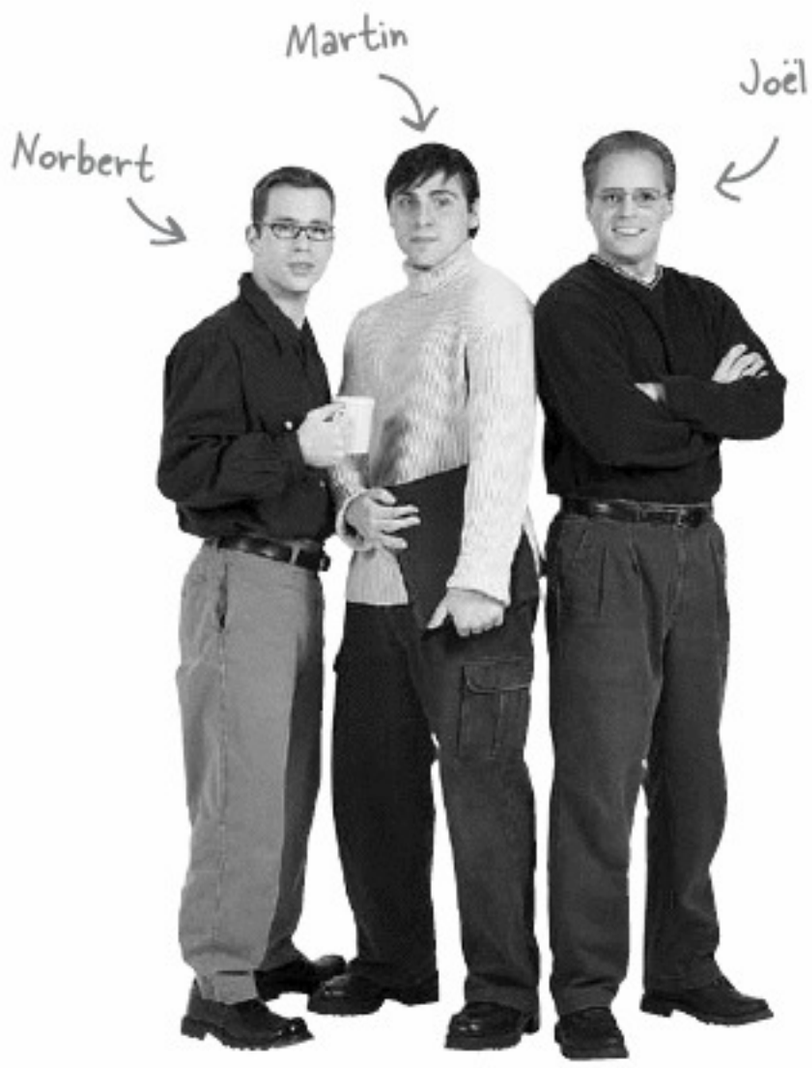
Par chance, vous pouvez devenir Greg...

Vous ÊTES Greg



Votre mission est de jouer une dernière fois le rôle de Greg et d'arranger le côté utilisateur de sa base de données pour que ses employés ne puissent pas y mettre la pagaille par accident.

Lisez les descriptions des emplois de chaque utilisateur et trouvez les instructions GRANT qui leur donnent les données dont ils ont besoin sans les laisser accéder à ce qu'ils ne doivent pas toucher.



Norbert : « Je suis en charge de trouver des correspondances entre des emplois et des offres d'emplois. Je n'entre jamais rien dans la base de données, bien que je supprime les offres d'emplois quand j'ai trouvé des correspondances ou que l'emploi est pris. Parfois j'ai aussi besoin de chercher des informations sur les contacts dans `mes_contacts`. »

Martin : « Je saisis toutes les nouvelles données dans toute la base de données. Je suis devenu vraiment bon en INSERT maintenant que je ne peux plus saisir de X pour sexe. Je mets aussi les données à jour. J'apprends à utiliser DELETE, mais jusqu'à présent Greg m'a demandé de ne pas le faire. Bien sûr, ce qu'il ne sait pas... »

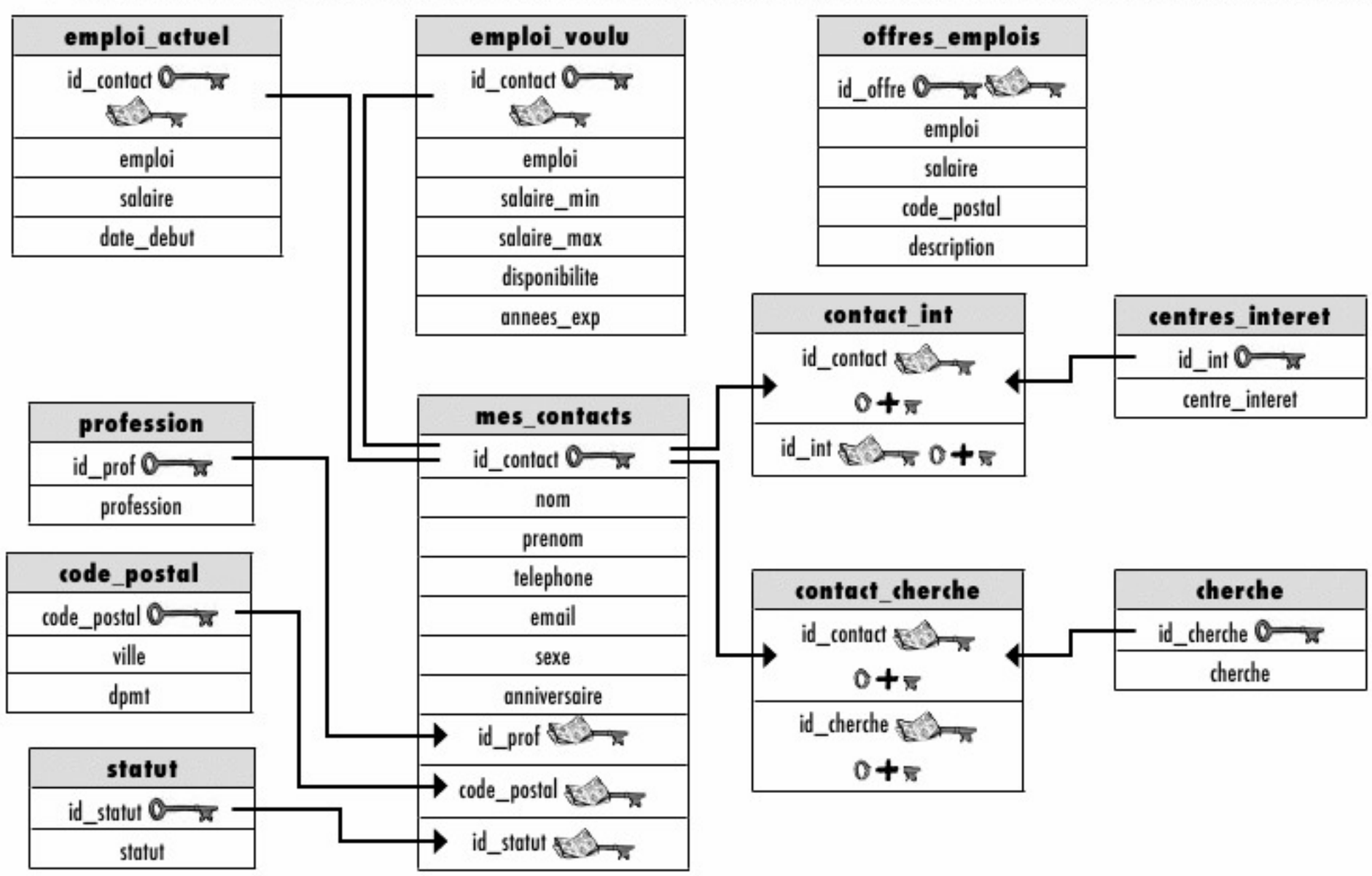
Joël : « Je viens d'être engagé par Greg pour gérer la dimension compatibilité des choses. Il veut intégrer ses infos de contacts dans un site web. Je suis plus un développeur web qu'un spécialiste SQL, mais je peux faire de simples SELECT. Je ne connais pas les INSERT. Ni Windows. Désolé, mauvaise blague. »

Regardez la base de données liste_greg et donnez des permissions à ces employés avant qu'ils ne saccagent des données.

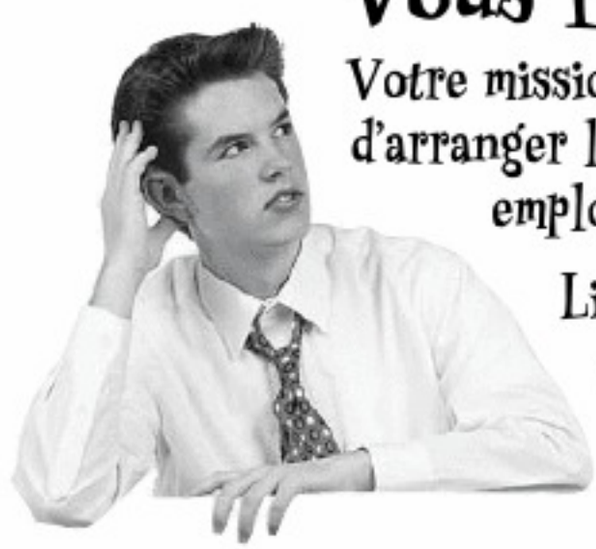
Écrivez une commande pour donner un mot de passe à l'utilisateur actuellement appelé « root ».

Écrivez trois commandes pour créer des comptes utilisateur pour chaque employé.

Écrivez des instructions GRANT pour donner les bonnes permissions à chaque nouvel employé.



Vous ÊTES Greg - SOLUTION



Votre mission est de jouer une dernière fois le rôle de Greg et d'arranger le côté utilisateur de sa base de données pour que ses employés ne puissent pas y mettre la pagaille par accident.

Lisez les descriptions des emplois de chaque utilisateur et trouvez les instructions GRANT qui leur donnent les données dont ils ont besoin sans les laisser accéder à ce qu'ils ne doivent pas toucher.

Écrivez une commande pour donner un mot de passe à l'utilisateur actuellement appelé « root ».

```
SET PASSWORD FOR root@localhost = PASSWORD('gr3GRulz');
```

Écrivez trois commandes pour créer des comptes utilisateur pour chaque employé.

```
CREATE USER norbert IDENTIFIED BY 'jObM4teH';
CREATE USER martin IDENTIFIED BY 'NOmOr3Xs';
CREATE USER joel IDENTIFIED BY 's3LeCTdOOd';
```

Ne vous inquiétez pas si vos mots de passe sont différents. Tant que vous avez les bons morceaux de commandes dans le bon ordre, c'est ce qui compte !

Écrivez des instructions GRANT pour donner les bonnes permissions à chaque nouvel employé.

```
GRANT DELETE ON offres_emplois TO norbert;
GRANT SELECT ON liste_greg.* TO norbert;
```

Norbert doit pouvoir enlever des offres et chercher (select) dans toutes les tables de liste_greg.

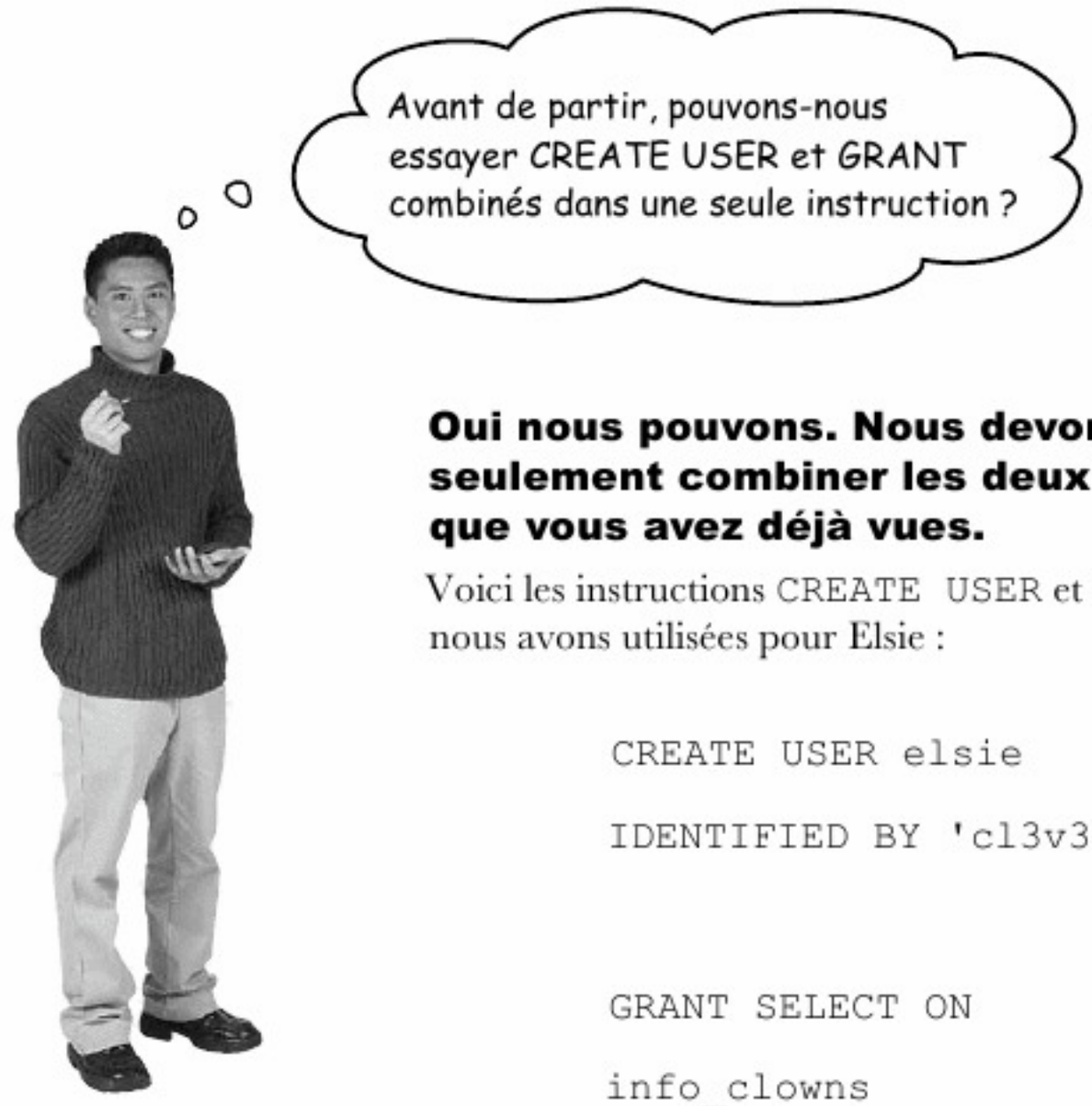
```
GRANT SELECT, INSERT ON liste_greg.* TO martin;
```

Martin a besoin d'un accès au SELECT et à l'INSERT pour toute la liste_greg. Pour l'instant, nous le gardons loin de DELETE.

```
GRANT SELECT ON mes_contacts, profession, code_postal, statut,
contact_int, centres_interet, contact_cherche, cherche TO joel;
```

Joël, lui, doit pouvoir sélectionner dans toutes les tables d'origine, mais pas celles qui concernent les emplois.

Combiner CREATE USER et GRANT



Oui nous pouvons. Nous devons seulement combiner les deux parties que vous avez déjà vues.

Voici les instructions CREATE USER et GRANT que nous avons utilisées pour Elsie :

```
CREATE USER elsie
IDENTIFIED BY 'cl3v3rp4s5w0rd';

GRANT SELECT ON
info_clowns
TO elsie;
```

Nous pouvons les combiner et abandonner la partie CREATE USER. Parce que l'utilisateur *elsie* doit être créé avant qu'on puisse lui accorder des privilèges, votre SGBDR vérifie qu'elle existe, et si ce n'est pas le cas, il crée son compte automatiquement.

```
GRANT SELECT ON
info_clowns
TO elsie
IDENTIFIED BY 'cl3v3rp4s5w0rd';
```

La Liste de Greg devient internationale !

Grâce à votre aide, Greg est maintenant tellement à l'aise pour utiliser SQL (et pour apprendre à Martin, Norbert et Joël à l'utiliser) qu'il a agrandi la Liste de Greg pour inclure les petites annonces locales ainsi que des forums.

Et la meilleure nouvelle ? C'est un tel succès à Dataville que plus de 500 villes dans le monde ont maintenant leurs propres Liste de Greg et Greg fait la première page des journaux !

HEBDO

ENQUÊTEUR

L'incroyable succès de la Liste de Greg

Franchises et Forums

Ses amis et parents disent que la célébrité n'a pas du tout changé Greg.

Par Fleur Delacour
JOURNALISTE À L'ENQUÊTEUR

DATAVILLE – Le chef d'entreprise local Greg connaît une réussite sensationnelle. Sa base de données communautaire est passée du stade de simples pense-bêtes à une table, puis à une base de données multi-tables qui permet de faire des rencontres, de trouver du travail, et bien plus.

Si vous voulez parler de jointures internes, de transactions et de privilèges avec des personnes qui vous ressemblent, ne cherchez pas plus loin que le forum SQL que vous trouverez ici :

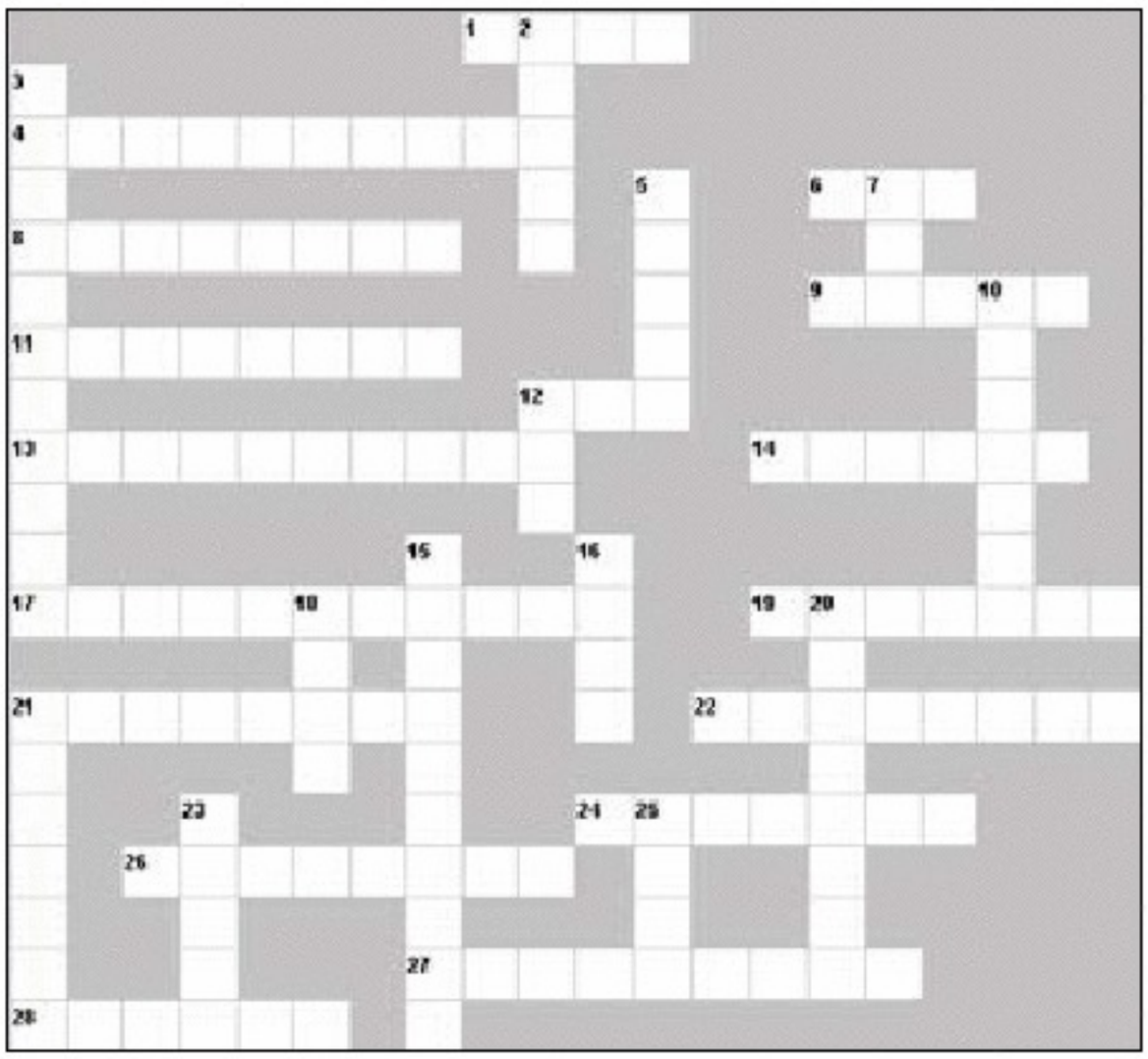
www.tetelapremiere.fr

Mais surtout, chers acrobates du SQL, amusez-vous !

La Liste de Greg a-t-elle déjà atteint votre ville ? Ce n'est qu'une question de temps, disent les analystes de données de la ville.

(les derniers) SQL-croisés

Oui, c'est un triste jour, vous avez sous les yeux les derniers mots-croisés du livre. Respirez largement, nous les avons bourrés de mots-clés et de commandes pour qu'ils durent plus longtemps. Amusez-vous !



Définitions horizontales :

1. Pour vous aider à décider des étapes de votre SQL qui peuvent être considérées comme des transactions, souvenez-vous de l'acronyme ... .

4. Une clé étrangère auto-... est la clé primaire d'une table utilisée dans cette même table dans un autre but.

6. Vous pouvez trouver la plus grande valeur d'une colonne avec cette fonction.

8. Les tables ... n'ont pas de données dupliquées, ce qui réduit la taille de votre base de données.

9. Accorder un rôle WITH ... OPTION permet à un utilisateur d'accorder ce rôle à quelqu'un d'autre.

11. Ceci signifie que vos données ont été divisées en leurs plus petites parties, qui ne peuvent ou ne doivent pas être divisées.

12. Utilisez cette clause dans votre instruction UPDATE pour modifier une valeur.

13. Une dépendance fonctionnelle ... signifie qu'une colonne non-clé est liée à une ou plusieurs autres colonnes non-clés.

14. La non équi-jointure renvoie toutes les lignes qui ne sont pas ... .

17. Si la sous-requête est indépendante et ne fait référence à rien de la requête externe, vous avez une sous-requête ... .

19. Utilisez ces deux mots pour classer vos résultats selon l'ordre alphabétique d'une colonne que vous précisez.

21. Une sous-requête ... signifie que la requête interne s'appuie sur la requête externe pour être résolue.

22. Cette fonction renvoie chaque valeur une seule fois, sans duplicats.

24. Cette jointure ne fonctionne que si les colonnes par lesquelles vous la faites ont le même nom dans les deux tables.

26. ... donne à l'utilisateur la permission d'utiliser SELECT, UPDATE, INSERT et DELETE dans la table spécifiée.

27. Une CLÉ ... est une CLÉ PRIMAIRE composée de plusieurs colonnes, créant une clé unique.

28. Une sous-requête est toujours une unique instruction ... .

Définitions verticales :

2. Une contrainte ... restreint les valeurs qui peuvent être insérées dans une colonne.

3. Pendant une ..., si toutes les étapes ne peuvent pas être réalisées sans interférence, aucune d'entre elles ne doit être réalisée.

5. Vous pouvez contrôler exactement ce que les utilisateurs peuvent faire dans les tables et les colonnes avec l'instruction ... .

7. On peut donner de nouvelles colonnes à notre table avec l'instruction ALTER et la clause ... COLUMN.

10. Vous ne pouvez avoir qu'un champ AUTO_ INCREMENT par table, ce doit être un type de données ... .

12. ... PASSWORD FOR 'root'@'localhost' = PASSWORD('b4dcl0wnZ');

15. Si modifier une des colonnes non-clés peut provoquer des changements sur les autres colonnes, vous avez une ... transitive.

16. Un ... OUTER JOIN prend toutes les lignes de la table de gauche et les fait correspondre à la table de DROITE.

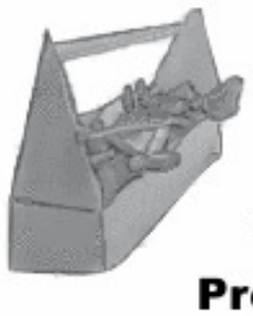
18. Assigner ceci est un moyen de regrouper des privilèges spécifiques et de les appliquer à tous les membres d'un groupe.

20. Utiliser ... quand vous voulez retirer des privilèges à un utilisateur renvoie une erreur si cet utilisateur a accordé ces privilèges à quelqu'un d'autre.

21. Les valeurs stockées dans des colonnes CHAR ou VARCHAR s'appellent comme ça.

23. Avec une jointure interne, vous comparez les lignes de deux tables, mais l'... de ces deux tables n'a pas d'importance.

25. Nous pouvons utiliser une ... jointure pour simuler deux tables.



Boîte à outils SQL

Félicitations, vous avez terminé le chapitre 12 ! Prenez une minute pour revoir les principes de sécurité SQL que nous venons de traiter. Pour une lsite exhaustive des outils du livre, voyez l'annexe C.

CREATE USER

Utilisé par certains SGBDR pour vous permettre de créer un utilisateur et de lui donner un mot de passe.

GRANT

Vous permet de contrôler exactement ce que les utilisateurs peuvent faire dans les tables et les colonnes, à partir des privilèges que vous leur donnez.

REVOKE

Utilisez cette instruction pour retirer des privilèges à un utilisateur.

WITH GRANT OPTION

Permet aux utilisateurs de donner à d'autres utilisateurs les privilèges qu'ils ont.

WITH ADMIN OPTION

Permet à quelqu'un ayant un rôle d'assigner ce rôle à quelqu'un d'autre.

Role

Un rôle est un groupe de privilèges. Les rôles vous permettent de regrouper des privilèges spécifiques et de les assigner à plus d'un utilisateur.

(les derniers) SQL-croisés Solution

								[1]A	[2]C	I	D									
[3]T									H											
[4]R	É	F	É	R	E	N	C	É	E											
A									C		[5]G				[6]M	[7]A	X			
[8]N	O	R	M	A	L	E	S		K		R					D				
S											A				[9]A	D	M	[10]I	N	
[11]A	T	O	M	I	Q	U	E				N							N		
C									[12]S	E	T							T		
[13]T	R	A	N	S	I	T	I	V	E					[14]É	G	A	L	E	S	
I									T									G		
O							[15]D			[16]L								E		
[17]N	O	N	C	O	[18]R	R	É	L	É	E				[19]O	[20]R	D	E	R	B	Y
					Ô		P			F					E					
[21]C	O	R	R	É	L	É	E			T		[22]D	I	S	T	I	N	C	T	
H					E		N								T					
A			[23]O				D			[24]N	[25]A	T	U	R	A	L				
Î		[26]G	R	A	N	T	A	L	L		U			I						
N			D				N				T			C						
E			R				[27]C	O	M	P	O	S	I	T	E					
[28]S	E	L	E	C	T		E													

Que diriez-vous d'une Liste de Greg dans vote ville ?

Utilisez SQL dans vos projets et vous aussi vous pourriez être comme Greg !

Ce fut un plaisir de vous recevoir ici à Dataville. Et nous sommes triste de vous voir partir, mais il n'y a rien de tel que de *réutiliser ce que vous avez appris* **dans vos propres bases de données** ; nous savons qu'il y a des clowns à espionner, des crêpes à tester ou des Listes de [notez votre nom ici] à créer, où que vous soyez. Quelques trésors vous attendent encore à la fin du livre, un index à parcourir, puis il sera temps de mettre toutes ces nouvelles idées en pratique. Nous sommes impatient de savoir comment vont vos affaires, alors ***envoyez-nous un message*** depuis le site de Tête la première, **www.tetelapremiere.fr**, pour nous dire ce que SQL **VOUS** a apporté !

Les dix sujets les plus importants (dont nous n'avons pas parlé)

Même après tout cela, il en reste encore un peu. Il y a juste quelques choses en plus que nous pensons que vous devez savoir. Cela nous gênerait de les ignorer, même si elles ne méritent qu'une simple évocation. Alors avant de reposer le livre, parcourez ces **courts, mais importants, petits bijoux du SQL**. Par ailleurs, une fois que vous aurez fini, il ne restera que deux annexes... et l'index... et peut-être un peu de pub... et alors vous aurez vraiment terminé. Promis !

#1. Une interface graphique pour votre SGBDR

Il est important de pouvoir coder votre SQL directement dans une console, mais maintenant vous savez ce que vous faites. Vous méritez un moyen plus simple de créer vos tables et d'en voir le contenu.

Tous les SGBDR ont un genre d'interface utilisateur graphique qui leur est associée. Voici un bref topo sur les outils d'interface graphique disponibles pour MySQL.

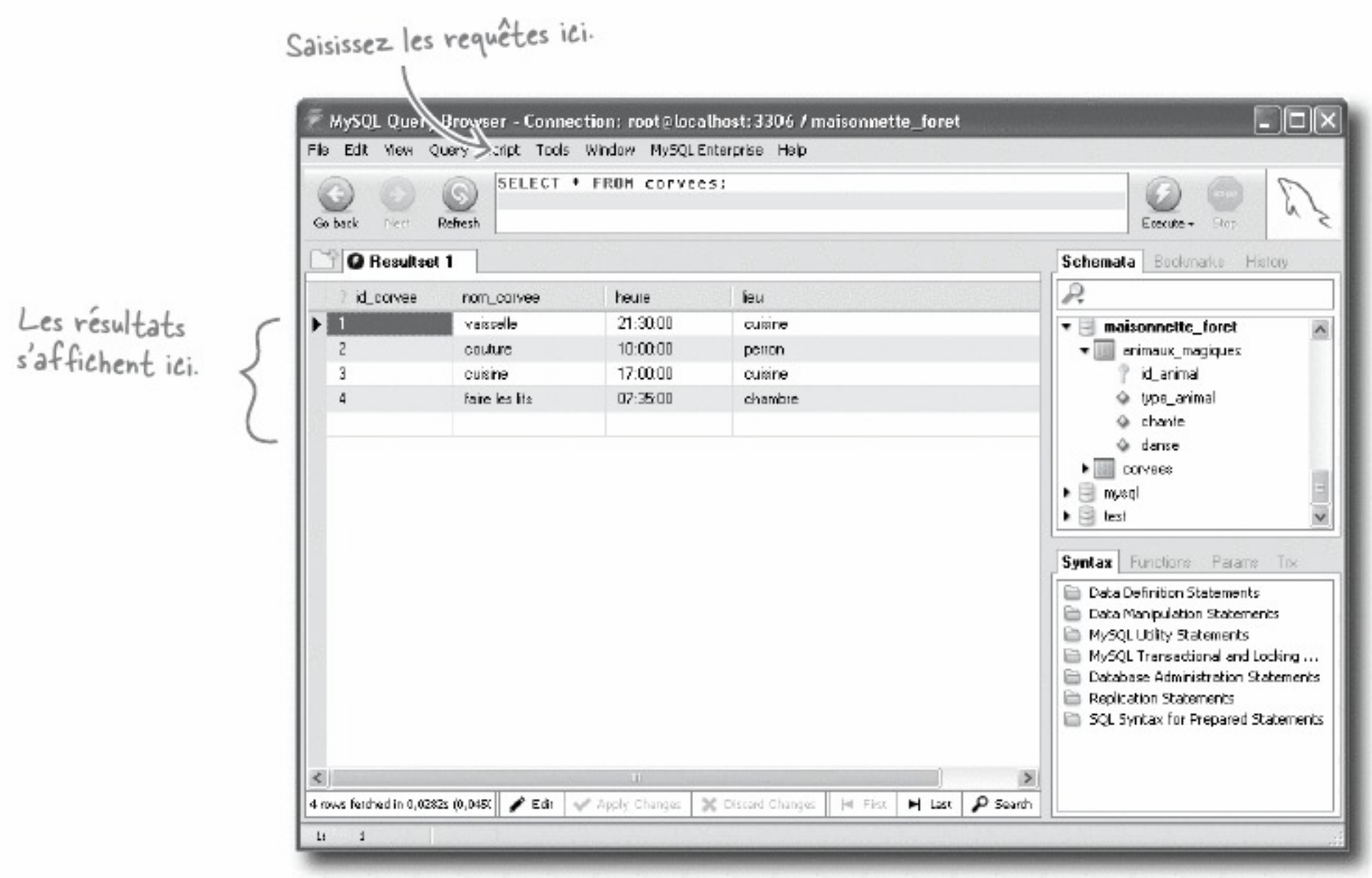
Outils d'interface graphique MySQL

Quand vous téléchargez MySQL, vous pouvez aussi télécharger les outils GUI de MySQL, et notamment MySQL Administrator. Vous pouvez les obtenir directement sur cette page :

`http://dev.mysql.com/downloads/gui-tools/5.0.html`

Ils sont disponibles pour Windows, Mac et Linux, mais uniquement en anglais. MySQL Administrator vous permet de voir, créer et modifier facilement vos tables et vos bases de données.

Vous aimerez aussi MySQL Query Browser. Vous pouvez y taper vos requêtes et voir les résultats dans l'interface du logiciel plutôt que dans une fenêtre de console.

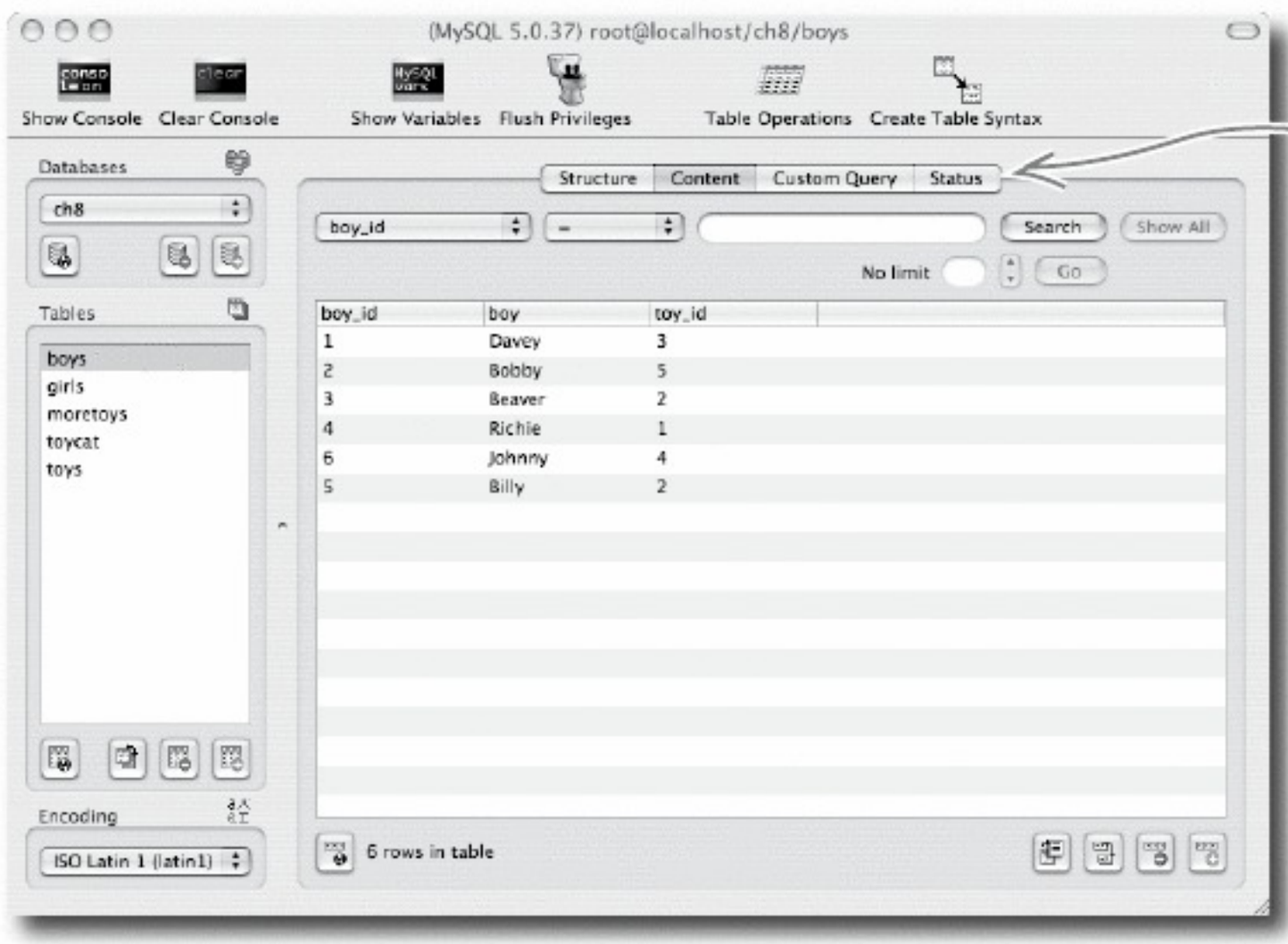


Autres outils GUI

Il existe quelques autres outils. Nous vous laissons choisir celui que vous préférez parmi ceux-ci. Il y en a beaucoup d'autres, non mentionnés ici, que vous pouvez facilement trouver avec une recherche sur internet.

Pour Mac, vous pouvez essayer CocoaMySQL :

http://cocoamysql.sourceforge.net/



Vous pouvez facilement voir la structure, exécuter une requête et modifier votre table avec ces boutons.

Si vous avez besoin d'une solution en ligne, essayez phpMyAdmin. Cela fonctionne bien si vous utilisez un compte hébergé en ligne avec MySQL sur un serveur web distant. Ce n'est pas aussi intéressant si vous utilisez votre ordinateur local. Vous pouvez trouver plus d'informations ici :

http://www.phpmyadmin.net/

Voici quelques autres outils communément utilisés. Certains sont réservés aux PC ; le mieux est de visiter les sites et de lire les informations sur leur dernière sortie pour voir s'ils vous conviennent :

Navicat offre un essai gratuit de 30 jours ici :

http://french.navicat.com/download.html

SQLyog offre une édition communautaire gratuite ici :

http://www.webyog.com/en/

#2. Mots réservés et caractères spéciaux

Le langage SQL est composé d'un certain nombre de mots-clés réservés. Il vaut mieux éviter ces mots dans les noms de vos bases de données, tables et colonnes. Si vous voulez appeler votre nouvelle table « select », essayez de trouver quelque chose de plus descriptif, qui n'utilise pas le mot « select ». Si vous devez utiliser un mot-clé réservé, essayez de l'utiliser avec d'autres mots et des tirets bas pour ne pas embrouiller votre SGBDR. Pour plus de commodité, vous trouverez sur la page de droite une liste de ces mots réservés que vous devez éviter dans vos noms:

Pour compliquer un peu plus les choses, SQL a une liste de mots non-réservés qui peuvent devenir réservés dans de futures versions de SQL. Nous n'allons pas en faire la liste ici, mais vous les trouverez dans le livre de référence de votre SGBDR que vous devriez acheter quand vous aurez fini ce livre.

Caractères spéciaux

Voici la liste de la plupart des caractères qu'utilise SQL et ce à quoi ils servent. Comme pour les mots réservés, il vaut mieux éviter de les utiliser dans vos noms, à l'exception du tiret bas (_), que nous vous encourageons à utiliser dans vos noms. En général, il vaut mieux éviter tout ce qui n'est pas une lettre non accentuée ou un tiret bas dans vos noms de tables. Les nombres ne sont pas non plus une très bonne idée, à moins qu'ils ne soient descriptifs d'une certaine façon.

*	Renvoie toutes les colonnes d'une table à partir d'une instruction a SELECT.
( )	Utilisé pour grouper des expressions , précise l'ordre dans lequel réaliser des opérations mathématiques et dans lequel faire des appels de fonctions. Egalement utilisé pour encadrer des sous-requêtes.
;	Termine vos instructions SQL.
,	Sépare les éléments d'une liste. Utilisé dans les instructions INSERT et les clauses IN.
.	Utilisé pour faire référence à des noms de tables et utilisé dans les nombres décimaux.
_	C'est un joker qui représente un caractère dans une clause LIKE.
%	Un autre joker de clause LIKE , celui-ci représente plusieurs caractères.
!	Le point d'exclamation remplace NOT. Il est utilisé pour les comparaisons dans les clauses WHERE.
'	Une paire de guillements simples indique à SQL qu'une chaîne se trouve entre les deux.
"	Vous pouvez aussi utiliser une paire de guillemets doubles de la même façon, même s'il vaut mieux garder les guillemets simples.
\	Vous utilisez ceci pour introduire une apostrophe dans une colonne de texte de votre table.
+	En plus de l'utiliser pour les additions, vous pouvez aussi l'utiliser pour joindre ou concaténer deux chaînes.

Ces signes ne sont des jokers que quand ils sont utilisés avec LIKE.

Voici un aperçu rapide des opérateurs mathématiques :

+	Addition	–	Soustraction	*	*Entre deux valeurs*, l'astérisque agit comme signe de multiplication	/	Division

Et les opérateurs de comparaison :

>	Supérieur à	!>	Non supérieur à	>=	Supérieur ou égal à
<	Inférieur à	!<	Non inférieur à	<=	Inférieur ou égal à
=	Egal à	<>	Différent de	!=	Non égal à

&	|	^	~
Non traités dans ce livre. Consultez le manuel de votre SGBDR pour plus d'infos..			

Mots réservés

Nous vous suggérons de les parcourir quand vous donnez un nom composé d'un seul mot à quelque chose pour vérifier que vous n'utilisez pas l'un d'eux.

A	ABSOLUTE ACTION ADD ADMIN AFTER AGGREGATE ALIAS ALL ALLOCATE ALTER AND ANY ARE ARRAY AS ASC ASSERTION AT AUTHORIZATION
B	BEFORE BEGIN BINARY BIT BLOB BOOLEAN BOTH BREADTH BY
C	CALL CASCADE CASCADED CASE CAST CATALOG CHAR CHARACTER CHECK CLASS CLOB CLOSE COLLATE COLLATION COLUMN COMMIT COMPLETION CONNECT CONNECTION CONSTRAINT CONSTRAINTS CONSTRUCTOR CONTINUE CORRESPONDING CREATE CROSS CUBE CURRENT CURRENT_DATE CURRENT_PATH CURRENT_ROLE CURRENT_TIME CURRENT_TIMESTAMP CURRENT_USER CURSOR CYCLE
D	DATA DATE DAY DEALLOCATE DEC DECIMAL DECLARE DEFAULT DEFERRABLE DEFERRED DELETE DEPTH DEREF DESC DESCRIBE DESCRIPTOR DESTROY DESTRUCTOR DETERMINISTIC DICTIONARY DIAGNOSTICS DISCONNECT DISTINCT DOMAIN DOUBLE DROP DYNAMIC
E	EACH ELSE END END_EXEC EQUALS ESCAPE EVERY EXCEPT EXCEPTION EXEC EXECUTE EXTERNAL
F	FALSE FETCH FIRST FLOAT FOR FOREIGN FOUND FROM FREE FULL FUNCTION
G	GENERAL GET GLOBAL GO GOTO GRANT GROUP GROUPING
H	HAVING HOST HOUR
I	IDENTITY IGNORE IMMEDIATE IN INDICATOR INITIALIZE INITIALLY INNER INOUT INPUT INSERT INT INTEGER INTERSECT INTERVAL INTO IS ISOLATION ITERATE
J	JOIN
K	KEY
L	LANGUAGE LARGE LAST LATERAL LEADING LEFT LESS LEVEL LIKE LIMIT LOCAL LOCALTIME LOCALTIMESTAMP LOCATOR
M	MAP MATCH MINUTE MODIFIES MODIFY MODULE MONTH
N	NAMES NATIONAL NATURAL NCHAR NCLOB NEW NEXT NO NONE NOT NULL NUMERIC
O	OBJECT OF OFF OLD ON ONLY OPEN OPERATION OPTION OR ORDER ORDINALITY OUT OUTER OUTPUT
P	PAD PARAMETER PARAMETERS PARTIAL PATH POSTFIX PRECISION PREFIX PREORDER PREPARE PRESERVE PRIMARY PRIOR PRIVILEGES PROCEDURE PUBLIC
Q	
R	READ READS REAL RECURSIVE REF REFERENCES REFERENCING RELATIVE RESTRICT RESULT RETURN RETURNS REVOKE RIGHT ROLE ROLLBACK ROLLUP ROUTINE ROW ROWS
S	SAVEPOINT SCHEMA SCROLL SCOPE SEARCH SECOND SECTION SELECT SEQUENCE SESSION SESSION_USER SET SETS SIZE SMALLINT SOME SPACE SPECIFIC SPECIFICTYPE SQL SQLEXCEPTION SQLSTATE SQLWARNING START STATE STATEMENT STATIC STRUCTURE SYSTEM_USER
T	TABLE TEMPORARY TERMINATE THAN THEN TIME TIMESTAMP TIMEZONE_HOUR TIMEZONE_MINUTE TO TRAILING TRANSACTION TRANSLATION TREAT TRIGGER TRUE
U	UNDER UNION UNIQUE UNKNOWN UNNEST UPDATE USAGE USER USING
V	VALUE VALUES VARCHAR VARIABLE VARYING VIEW
W	WHEN WHENEVER WHERE WITH WITHOUT WORK WRITE
X	
Y	YEAR
Z	ZONE

#3. ALL, ANY et SOME

Trois de ces mots-clés sont très pratiques dans les sous-requêtes : ALL, ANY et SOME. Ils fonctionnent avec des opérateurs de comparaison et des ensembles de résultats. Avant d'en parler, retournons en arrière jeter un œil à l'opérateur IN dont nous avons parlé dans le chapitre 9 :

notes_restaurants

nom	note
La maison des Pizzas	3
La Cabane	7
Chez Arthur	9
Côtelettes et cie	5

```
SELECT nom, note FROM notes_restaurants
WHERE note IN
(SELECT note FROM notes_restaurants
WHERE note > 3 AND note < 9);
```

Cette sous-requête renvoie toutes les notes entre 3 et 9 ; dans ce cas, 7 et 5.

Cette requête renvoie le nom de tout restaurant ayant la même note que le résultat de notre sous-requête dans l'ensemble entre parenthèses. Nos résultats sont : ***La Cabane*** et ***Côtelettes et cie***.

Utiliser ALL

Prenez maintenant cette requête :

```
SELECT nom, note FROM notes_restaurants
WHERE note > ALL
(SELECT note FROM notes_restaurants
WHERE note > 3 AND note < 9);
```

Cette fois nous allons récupérer tous les restaurants avec une note supérieure à toutes les notes de notre ensemble. Notre résultat est ***Chez Arthur***.

Voici une requête avec < :

```
SELECT nom, note FROM notes_restaurants
WHERE note < ALL
(SELECT note FROM notes_restaurants WHERE note >
3 AND note < 9);
```

Cette requête nous renvoie ***La maison des Pizzas***. Nous pouvons aussi utiliser >= et <= avec ALL. Avec la requête suivante, nous obtenons les notes supérieures à notre ensemble, ainsi que les notes égales au plus grand résultat de notre ensemble, qui est 7 :

```
SELECT nom, note FROM notes_restaurants
WHERE note >= ALL
(SELECT note FROM notes_restaurants
WHERE note > 3 AND note < 9);
```

Toutes les valeurs supérieures à notre ensemble ou égales au plus grand résultat de notre ensemble seront renvoyées.

Supérieur à ALL trouve toutes les valeurs plus grandes que la plus grande valeur de l'ensemble.

Inférieur à ALL trouve toutes les valeurs plus petites que la plus petite valeur de l'ensemble.

Utiliser ANY

ANY renvoie vrai si UN élément de notre ensemble remplit la condition. Prenez l'exemple suivant :

```
SELECT nom, note FROM notes_restaurants
WHERE note > ANY
(SELECT note FROM notes_restaurants WHERE
note > 3 AND note < 9);
```

Nous pouvons lire ceci comme : sélectionne toutes les lignes où la note est supérieure à l'un des éléments de l'ensemble (5, 7). Comme ***La Cabane*** a une note de 7, ce qui est supérieur à 5, elle est renvoyée. Et ***Chez Arthur*** avec une note de 9 est aussi renvoyé.

Supérieur à ANY trouve toutes les valeurs supérieures à la plus petite valeur de l'ensemble.

Inférieur à ANY trouve toutes les valeurs plus petites que la plus grande valeur de l'ensemble.

Utiliser SOME

SOME signifie la même chose que ANY en syntaxe standard de SQL et sous MySQL. Consultez votre version de SGBDR pour vérifier qu'il fonctionne de la même façon pour vous.

#4. Plus d'infos sur les types de données

Vous connaissez la plupart des types de données les plus courants, mais certains détails peuvent vous aider à affiner encore plus vos choix. Regardons de plus près quelques nouveaux types ainsi que certains que vous avez déjà utilisés.

BOOLEAN

Le type booléen vous permet de stocker 'true', 'false', ou peut être laissé `NULL`. C'est parfait pour toutes sortes de colonnes vrai/faux. En coulisses, votre SGBDR stocke un 1 pour les valeurs vraies et un 0 pour les valeurs fausses. Vous pouvez insérer 1 ou 'true', 0 ou 'false'.

INT

Nous avons utilisé `INT` dans ce livre. `INT` peut contenir **des valeurs entre 0 et 4294967295**. Ceci si vous ne voulez utiliser que des valeurs positives, et c'est ce que l'on appelle un ***entier non signé***.

Si vous voulez utiliser des valeurs positives et négatives dans votre integer, vous devez en faire un entier signé. Un entier signé peut contenir des valeurs entre –2147483648 et 2147483647. Pour dire à votre SGBDR que vous voulez que votre `INT` soit signé, utilisez cette syntaxe quand vous le créez:

```
INT(SIGNED)
```

Autres types INT

Vous connaissez déjà `INT`, mais les deux types `SMALLINT` et `BIGINT` l'affinent un peu. Ils précisent le nombre maximum qui peut être stocké.

L'éventail des valeurs qu'ils peuvent stocker varie selon votre SGBDR. Pour MySQL, les valeurs sont :

	signé	non signé
SMALLINT	-32768 à 32767	0 à 65535
BIGINT	-9223372036854775808 à 9223372036854775807	0 à 18446744073709551615

MySQL va un peu plus loin et ajoute ces types également :

	signé	non signé
TINYINT	-128 à 127	0 à 255
MEDIUMINT	-8388608 à 8388607	0 à 16777215

Les types DATE et TIME

Voici une synthèse des formats dans lesquels MySQL stocke vos types de données date et time :

DATE	**AAAA-MM-JJ**
DATETIME	**AAAA-MM-JJ HH:MM:SS**
TIMESTAMP	**AAAAMMJJHHMMSS**
TIME	**HH:MM:SS**

des_dates

une_date
2007-08-25 22:10:00
1925-01-01 02:05:00

Quand vous `SELECTionnez` un type date ou time, vous pouvez modifier ce que votre SGBDR renvoie. Les fonctions pour le faire varient selon les SGBDR. Voici un exemple de la fonction de MySQL `DATE_FORMAT()`

Les chaînes de format doivent être entre guillemets.

Supposez que vous ayez la colonne `une_date` :

```
SELECT DATE_FORMAT(une_date, '%M %Y') FROM des_dates;
```

Les signes `%M` et `%Y` indiquent à la fonction comment vous voulez formater les dates. Vos résultats ressembleraient à cela :

une_date
August 2007
January 1925

Nous n'avons pas la place ici d'entrer dans toutes les options de formattage ; elles sont très nombreuses. Mais grâce à elles, vous pouvez obtenir ce dont vous avez besoin dans vos champs date et time, sans devoir afficher ce dont vous n'avez pas besoin.

#5. Tables temporaires

Nous avons créé beaucoup de tables dans ce livre. À chaque fois que nous créons une table, notre SGBDR stocke la structure de cette table. Quand nous y insérons des données, ces données sont stockées. La table et ses données sont sauvegardées. Si vous fermez votre session SQL dans votre fenêtre de terminal ou votre interface graphique, cette table et ses données vont continuer à exister. Les données perdurent jusqu'à ce que vous les supprimiez ; la table, jusqu'à ce que vous lui envoyiez un drop.

SQL offre un autre type de table, appelée une ***table temporaire***. Une table temporaire existe à partir du moment où vous la créez jusqu'à ce que vous lui envoyiez un drop, ou *jusqu'à la fin de la session utilisateur*. Par **session** nous désignons le moment où vous vous connectez à votre compte jusqu'à ce que vous vous déconnectiez ou que vous fermiez votre programme. Vous pouvez aussi la supprimer explicitement avec l'instruction `DROP`.

Raisons de vouloir une table temporaire :

- Vous pouvez l'utiliser pour conserver des résultats intermédiaires, par exemple, exécuter sur une colonne des opérations mathématiques dont les résultats vous seront utiles pendant votre session actuelle, mais pas la suivante.
- Vous voulez une image du contenu d'une table à un moment précis.
- Vous souvenez-vous de notre conversion de la Liste de Greg d'une table en plusieurs ? Vous pouvez créer des tables temporaires pour vous aider à restructurer vos données et savoir que ces tables vont disparaître à la fin de votre session.
- Si vous utilisez SQL avec un langage de programmation un jour, vous pouvez créer des tables temporaires pour collecter des données, puis stocker les résultats définitifs dans une table normale.

Créer une table temporaire

Pour créer une table temporaire en MySQL, la syntaxe est simple ; vous ajoutez le mot-clé `TEMPORARY` :

```
CREATE TEMPORARY TABLE ma_table_temp
(
   une_id INT,
   une_donnee VARCHAR(50)
)
```

Nous avons seulement besoin d'ajouter le mot TEMPORARY.

Ouvrez les yeux !

La syntaxe de la création de tables temporaires varie beaucoup selon les SGBDR

Vérifiez-la dans la documentation de votre SGBDR.

Un raccourci pour table temporaire

Vous pouvez créer votre table temporaire à partir d'une requête comme :

```
CREATE TEMPORARY TABLE ma_table_temp AS
SELECT * FROM ma_table_normale;
```

N'importe quelle requête peut aller après AS.

#6. Lancez CAST sur vos données

Parfois vous avez un type de données dans une colonne, mais vous voulez afficher ces données avec un autre type de données. SQL a une fonction appelée `CAST()` qui peut convertir des données d'un type en un autre type.

La syntaxe est :

```
CAST(votre_colonne AS TYPE)
```

Le `TYPE` peut être au choix :

```
CHAR()
DATE
DATETIME
DECIMAL
SIGNED [INTEGER]
TIME
UNSIGNED [INTEGER]
```

Quelques situations où CAST() est utile

Convertir une chaîne avec une date en un type DATE :

```
SELECT CAST('2005-01-01' AS DATE);
```

La chaîne '2005-01-01' est formatée comme une DATE.

Convertir un entier en décimal :

```
SELECT CAST(2 AS DECIMAL);
```

L'entier 2 devient le décimal 2.00.

Vous pouvez notamment utiliser `CAST()` dans la liste des valeurs d'une instruction `INSERT` et à l'intérieur de la liste des colonnes d'un `SELECT`.

Vous ne pouvez pas utiliser CAST() dans ces situations

* Décimal vers entier

* `TIME`, `DATE`, `DATETIME`, `CHAR` vers `DECIMAL` ou `INTEGER`.

#7. Qui êtes-vous ? Quelle heure est-il ?

Parfois vous pouvez avoir plus d'un compte utilisateur sur votre SGBDR, chacun avec différents rôles et différentes permissions. Si vous avez besoin de savoir quel compte vous utilisez actuellement, cette commande vous le dira :

```
SELECT CURRENT_USER;
```

Elle vous dira qussi quelle est la machine qui vous héberge. Si votre SGBDR est sur le même ordinateur que vous et que vous utilisez le compte root, vous verrez :

```
root@localhost
```

Vous pouvez obtenir la date et l'heure actuelle avec ces commandes :

```
Fichier Edition Fenêtre Aide
> SELECT CURRENT_DATE;
+--------------+
| CURRENT_DATE |
+--------------+
| 2007-07-26   |
+--------------+
1 row in set (0.00 sec)
```

```
Fichier Edition Fenêtre Aide
> SELECT CURRENT_TIME;
+--------------+
| CURRENT_TIME |
+--------------+
| 11:26:48     |
+--------------+
1 row in set (0.00 sec)
```

```
Fichier Edition Fenêtre Aide
SELECT CURRENT_USER;
+----------------+
| CURRENT_USER   |
+----------------+
| root@localhost |
+----------------+
1 row in set (0.00 sec)
```

#8. Fonctions numériques utiles

Voici un résumé des fonctions qui fonctionnent avec les types de données numériques. Vous en avez déjà vu certaines :

fonction numérique	que fait-elle ?	
ABS(x)	Renvoie la valeur absolue de x	
	requête	**résultat**
	SELECT ABS(-23);	23
ACOS(x)	Renvoie l'arc cosinus de x	
	SELECT ACOS(0);	1.5707963267949
ASIN()	Renvoie l'arc sinus de x	
	SELECT ASIN(0.1);	0.10016742116156
ATAN(x,y)	Renvoie l'arc tangente de x et y	
	SELECT ATAN(-2,2);	-0.78539816339745
CEIL(x)	Renvoie le plus petit entier supérieur ou égal à x. La valeur renvoyée sera un BIGINT.	
	SELECT CEIL(1.32);	2
COS(x)	Revoie le cosinus de x en radians	
	SELECT COS(1);	0.54030230586814
COT(x)	Renvoie la cotangente de x	
	SELECT COT(12);	-1.5726734063977
EXP(x)	Renvoie le nombre d'Euler à la puissance x	
	SELECT EXP(-2);	0.13533528323661
FLOOR(x)	Renvoie le plus grand entier inférieur ou égal à x	
	SELECT FLOOR(1.32);	1
FORMAT(x,y)	Convertit x en chaîne formattée arrondie à y décimales	
	SELECT FORMAT(3452100.50,2);	3,452,100.50
LN(x)	Renvoie le logarithme naturel de x	
	SELECT LN(2);	0.69314718055995
LOG(x) and LOG(x,y)	Renvoie le logarithme naturel de x, ou avec deux paramètres, renvoie le log de x en base y	
	SELECT LOG(2);	0.69314718055995
	SELECT LOG(2,65536);	16

→ À suivre page suivante.

#8. Fonctions numériques utiles (suite)

fonction numérique	que fait-elle ?	
MOD(x,y)	Renvoie le reste de x divisé par y	
	requête	**résultat**
	SELECT MOD(249,10);	9
PI()	Renvoie la valeur de pi	
	SELECT PI();	3.141593
POWER(x,y)	Renvoie la valeur de x à la puissance y	
	SELECT POW(3,2);	9
RADIANS(x)	Renvoie x converti de degrés en radians	
	SELECT RADIANS(45);	0.78539816339745
RAND()	Renvoie une valeur aléatoire à virgule flottante	
	SELECT RAND();	0.84655920681223
ROUND(x)	Renvoie la valeur def x arrondie à l'entier le plus proche	
	SELECT ROUND(1.34);	1
	SELECT ROUND(-1.34);	-1
ROUND(x,y)	Renvoie la valeur de x arrondie à y chiffres après la virgule	
	SELECT ROUND(1.465, 1);	1.5
	SELECT ROUND(1.465, 0);	1
	SELECT ROUND(28.367, -1);	30
SIGN(x)	Renvoie 1 si x est positif, 0 si x est 0 ou -1 si x est négatif	
	SELECT SIGN(-23);	-1
SIN(x)	Renvoie le sinus de x	
	SELECT SIN(PI());	1.2246063538224e-16
SQRT(x)	Renvoie la racine carrée de x	
	SELECT SQRT(100);	10
TAN(x)	Renvoie la tangente de x	
	SELECT TAN(PI());	-1.2246063538224e-16
TRUNCATE(x,y)	Renvoie le nombre x tronqué à y décimales	
	SELECT TRUNCATE(8.923,1);	8.9

#9. Indexer pour accélérer les choses

Vous savez tout sur les index clé primaire et clé étrangère. Ces types d'index sont pratiques pour lier plusieurs tables entre elles et forcer l'intégrité des données. Mais vous pouvez aussi créer des index sur vos colonnes pour accélérer vos requêtes.

Quand un `WHERE` est exécuté sur une colonne non-indexée, le SGBDR commence au début de cette colonne et la parcourt une ligne à la fois. Si votre table est immense, et par immense nous entendons immense de 4 millions de lignes, cela peut commencer à prendre un temps perceptible.

Si vous créez un index sur une colonne, votre SGBDR garde des informations supplémentaires sur cette colonne, qui accélèrent énormément la recherche. Les informations supplémentaires sont gardées dans une table en coulisse triée dans l'ordre précis que le SGBDR peut parcourir le plus rapidement. La contrepartie est que les index prennent de la place. Vous pouvez donc envisager de créer certaines colonnes comme index, celles que vous interrogez le plus souvent, et de ne pas en indexer d'autres.

Voici le code `ALTER` table pour ajouter un index à une colonne :

```
ALTER TABLE mes_contacts
ADD INDEX (nom);
```

Il y a un peu plus de théorie derrière l'indexation, mais c'est l'idée générale.

#10. 2 minutes de PHP/MySQL

Avant de partir, jetez un très rapide coup d'œil à la façon dont PHP et MySQL peuvent interagir pour vous aider à récupérer vos données en ligne. Ce n'est qu'un mini-aperçu de ce que vous pouvez faire et vous devriez certainement en lire plus sur ce sujet.

Cet exemple assume que vous êtes quelque peu familier avec PHP. Et nous savons que vous connaissez bien l'écriture de requêtes à présent. Le code ci-dessous se connecte à une base de données appelée `liste_greg` et sélectionne tous les prénoms et noms des gens de la table `mes_contacts`. Le code PHP stocke toutes les données de la base de données dans une collection. La dernière partie du code affiche tous les prénoms et noms sur une page web :

```
<?php
$connexion = mysql_connect("localhost","greg","gr3gzpAs");
if (!$connexion)
  {
    die('Connexion impossible : ' . mysql_error());
  }

mysql_select_db("liste_greg", $connexion);

$resultat = mysql_query("SELECT prenom, nom FROM mes_contacts");

while($ligne = mysql_fetch_array($resultat))
  {
    echo $ligne['prenom'] . " " . $ligne['nom'];
    echo "<br />";
  }

mysql_close($connexion);
?>
```

Nous allons sauvegarder ce fichier sous le nom `nomsgreg.php` sur un serveur.

Chaque ligne vue de plus près

```
<?php
```

Cette première ligne indique au serveur que du code PHP va suivre.

```
$connexion = mysql_connect("localhost","greg","gr3gzpAs");
```

Pour nous connecter à `liste_greg`, nous devons dire au serveur où est situé notre SGBDR, quel est notre nom d'utilisateur et quel est notre mot de passe. Nous créons une chaîne de connexion avec ces informations, que nous appelons `$connexion`. La fonction PHP `mysql_connect()` prend ces infos et s'adresse à notre SGBDR pour voir s'ils peuvent communiquer.

```
if (!$connexion)
  {
    die('Connexion impossible : ' . mysql_error());
  }
```

Si cela ne fonctionne pas, PHP va nous envoyer un message nous disant pourquoi il n'a pas pu se connecter au SGBDR et le PHP va arrêter son exécution.

```
mysql_select_db("liste_greg", $connexion);
```

D'accord, alors notre connexion au SGBDR fonctionne. Nous devons maintenant indiquer à PHP la base de données qui nous intéresse. Nous voulons utiliser notre base de données préférée, `liste_greg`.

```
$resultat = mysql_query("SELECT prenom, nom FROM mes_contacts");
```

Notre base de données est sélectionnée et nous sommes connecté, mais nous n'avons pas de requête. Nous en écrivons une et utilisons la fonction `mysql_query()` pour l'envoyer au SGBDR. Toutes les lignes renvoyées sont stockées dans une collection appelée `$resultat`.

```
while($ligne = mysql_fetch_array($resultat))
  {
```

Nous utilisons maintenant PHP pour récupérer toutes ces lignes dans `$resultat` et les envoyer vers la page web. C'est une petite boucle qui fait cela : elle passe dans une ligne à la fois, jusqu'à ce qu'elle atteigne la fin des données.

```
    echo $ligne['prenom'] . " " . $ligne['nom'];
    echo "<br />";
  }
```

Ces instructions `echo` en PHP écrivent le prénom et le nom de chaque ligne sur la page web. Un marqueur HTML `<br>` est inséré entre chaque ligne.

```
mysql_close($connexion);
```

Quand nous avons fini d'écrire tous les noms, nous fermons la connexion au SGBDR. C'est comme fermer le terminal.

```
?>
```

Enfin, nous terminons le script PHP.

Voyez par vous-même

Tous vos nouveaux talents en SQL ne vous apporteront pas grand chose sans un endroit où les exercer. Cette annexe contient les instructions pour installer votre propre SGBDR MySQL et pouvoir travailler dessus.

Pouvoir commencer, vite !

Parce que ce n'est pas très amusant d'avoir un livre sur SQL sans pouvoir l'essayer soi-même, voici une brève introduction pour installer MySQL sous Windows et Mac OS X.

NOTE : Cette section traite de Windows 2000, XP, Windows Server 2003 ou tout autre système d'exploitation Windows de 32-bit. Pour Mac, elle s'applique à Mac OS X 10.3.x ou plus récent.

Nous allons vous présenter le téléchargement et l'installation de MySQL. Ces temps-ci, le nom officiel de la version gratuite du serveur SGBDR MySQL est **MySQL Community Server**.

Instructions et dépannage

Dans la suite de cette annexe, vous trouverez une liste d'étapes pour installer MySQL sous Windows et Mac OS X. Ce n'est *pas* prévu pour remplacer les excellentes instructions présentes sur le site MySQL et **nous vous recommandons fortement d'aller les lire !** Pour des instructions plus détaillées, ainsi qu'un guide de dépannage, rendez-vous sur :

Prenez la version 5.0 ou la plus récente.

`http://dev.mysql.com/doc/refman/5.0/fr/windows-installation.html`

Vous apprécierez aussi MySQL Query Browser dont nous avons parlé aux pages 526–527. Vous pouvez y taper vos requêtes et en voir les résultats dans l'interface du logiciel plutôt que dans une fenêtre de console.

Étapes pour installer MySQL sous Windows

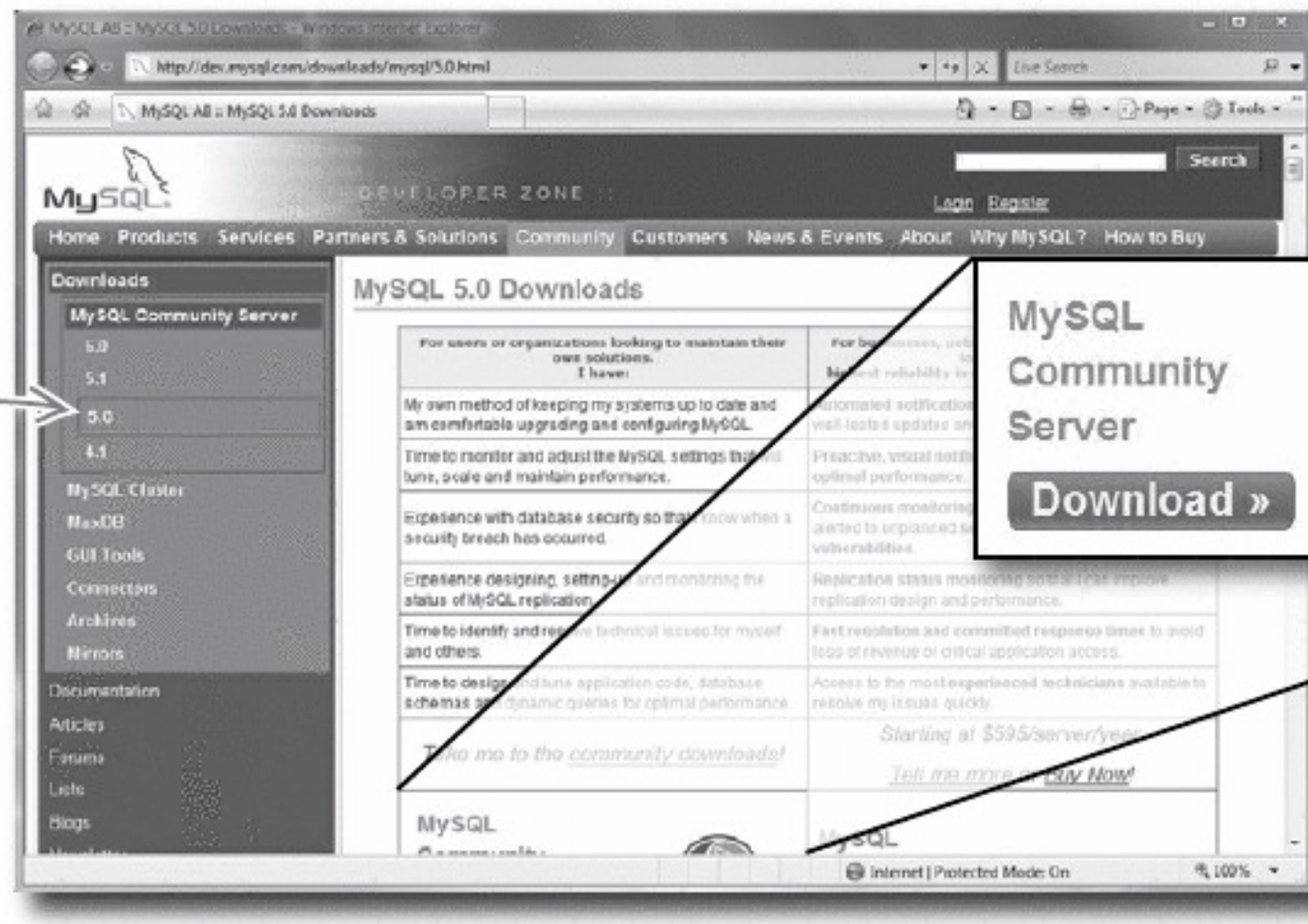
1 Allez sur :

`http://dev.mysql.com/downloads/mysql/5.0.html`

et cliquez sur le bouton de téléchargement de MySQL Community Server.

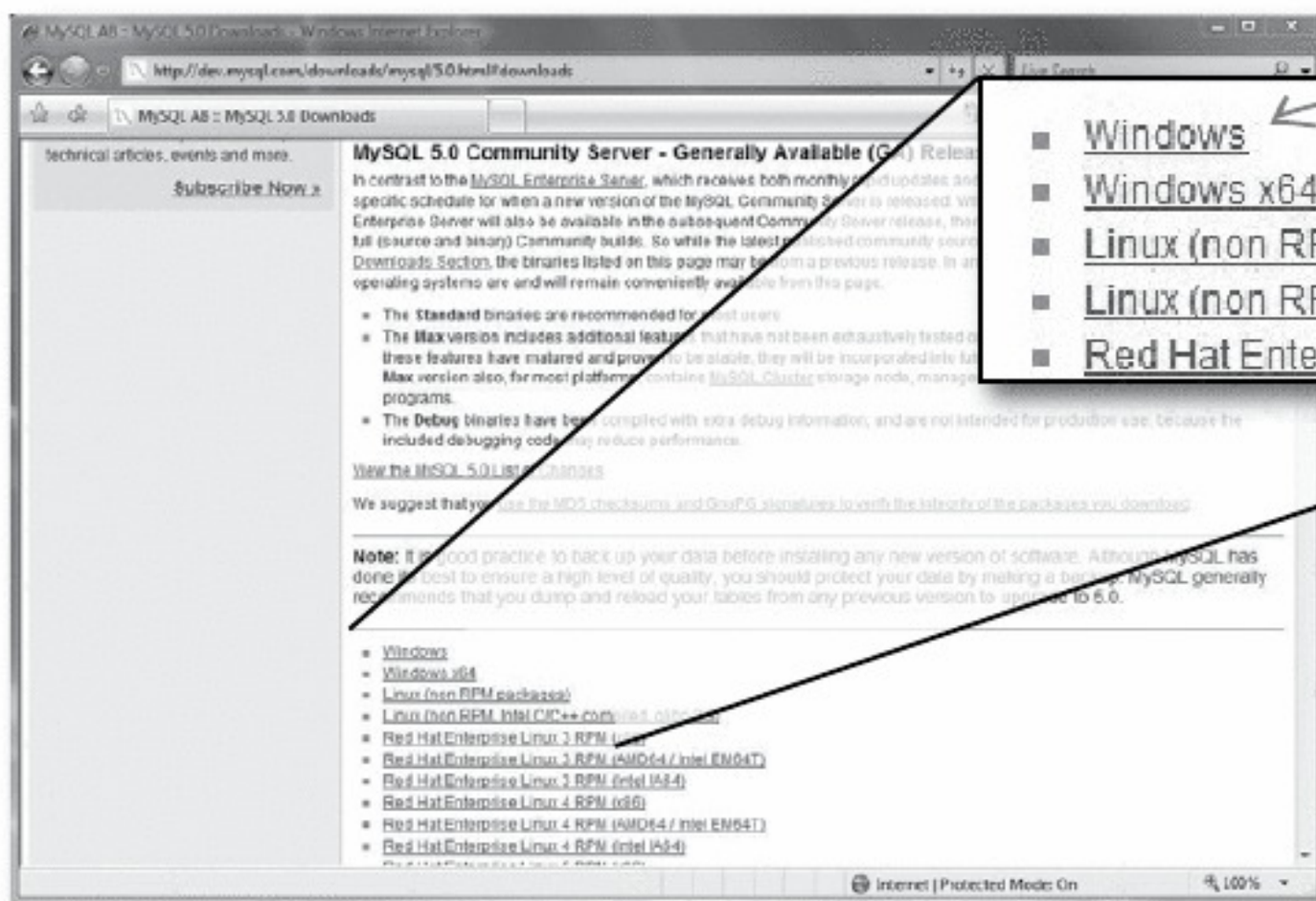
Vous devrez peut-être faire défiler un peu votre écran vers le bas.

Prenez la version 5.0 ou plus récente.



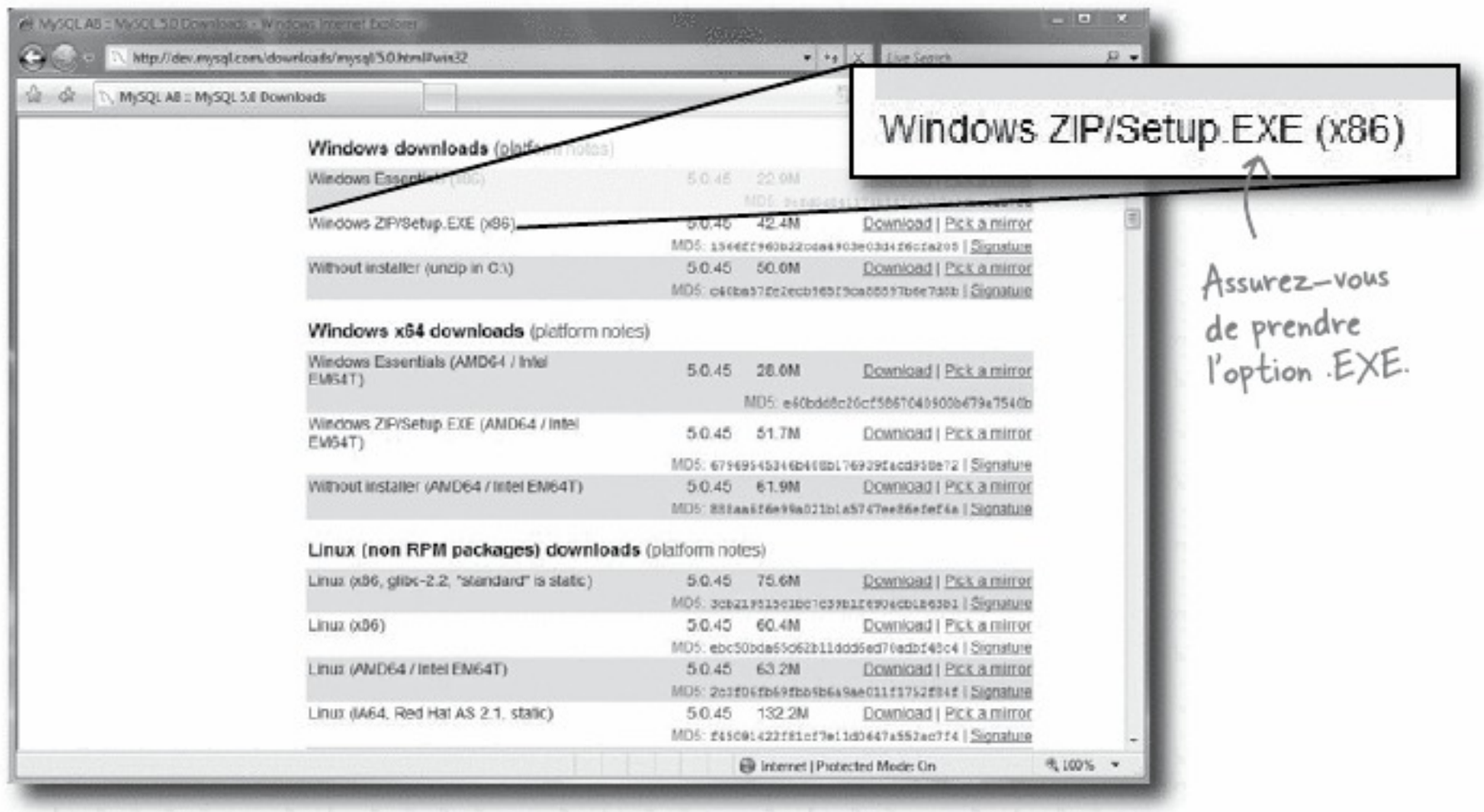
2 Dans la liste, choisissez **Windows**.

Le premier !



Télécharger votre installateur

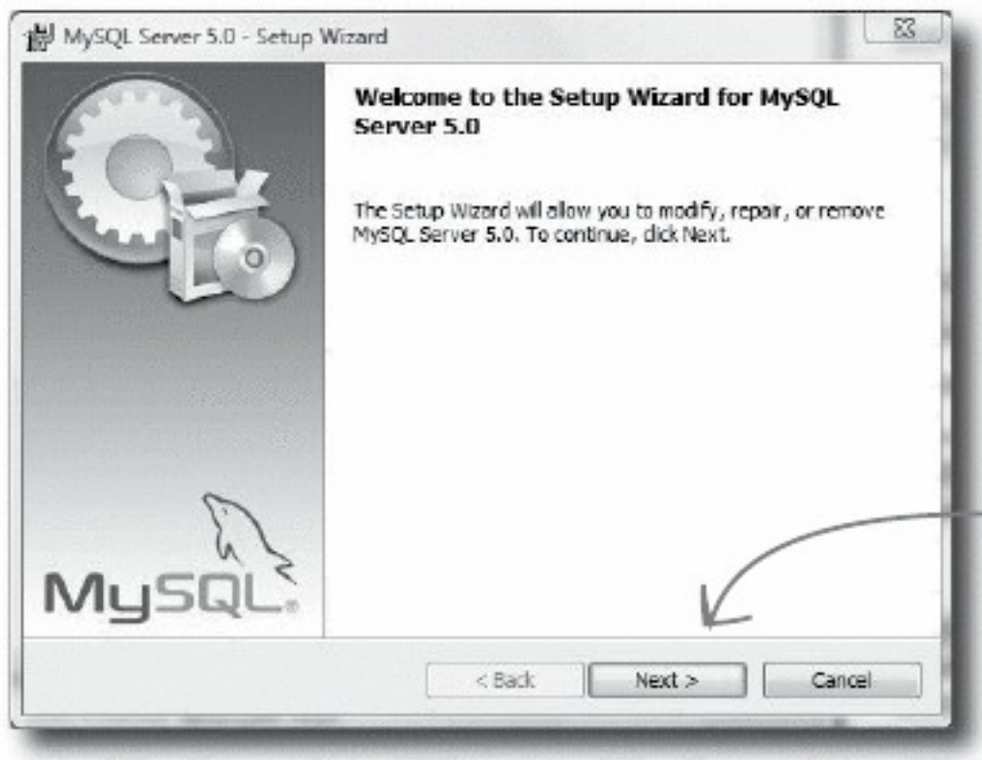
3 Dans **Windows downloads**, nous vous recommandons de choisir l'option `Windows ZIP/Setup.EXE` parce qu'elle contient un installateur qui simplifie beaucoup l'installation. Cliquez sur **Pick a Mirror**.



Assurez-vous de prendre l'option .EXE.

4 Vous verrez une liste de serveurs qui possèdent une copie que vous pouvez télécharger ; choisissez celui qui est le plus proche de vous (HTTP).

5 Quand le fichier est téléchargé et dézippé, double-cliquez sur **Setup.exe** pour le lancer. À partir de là, vous serez guidé dans l'installation par le **Setup Wizard**. Cliquez sur le bouton ***Next***.



Quand vous avez double-cliqué sur le fichier et que le dialogue du Setup Wizard apparaît, cliquez sur le bouton Next.

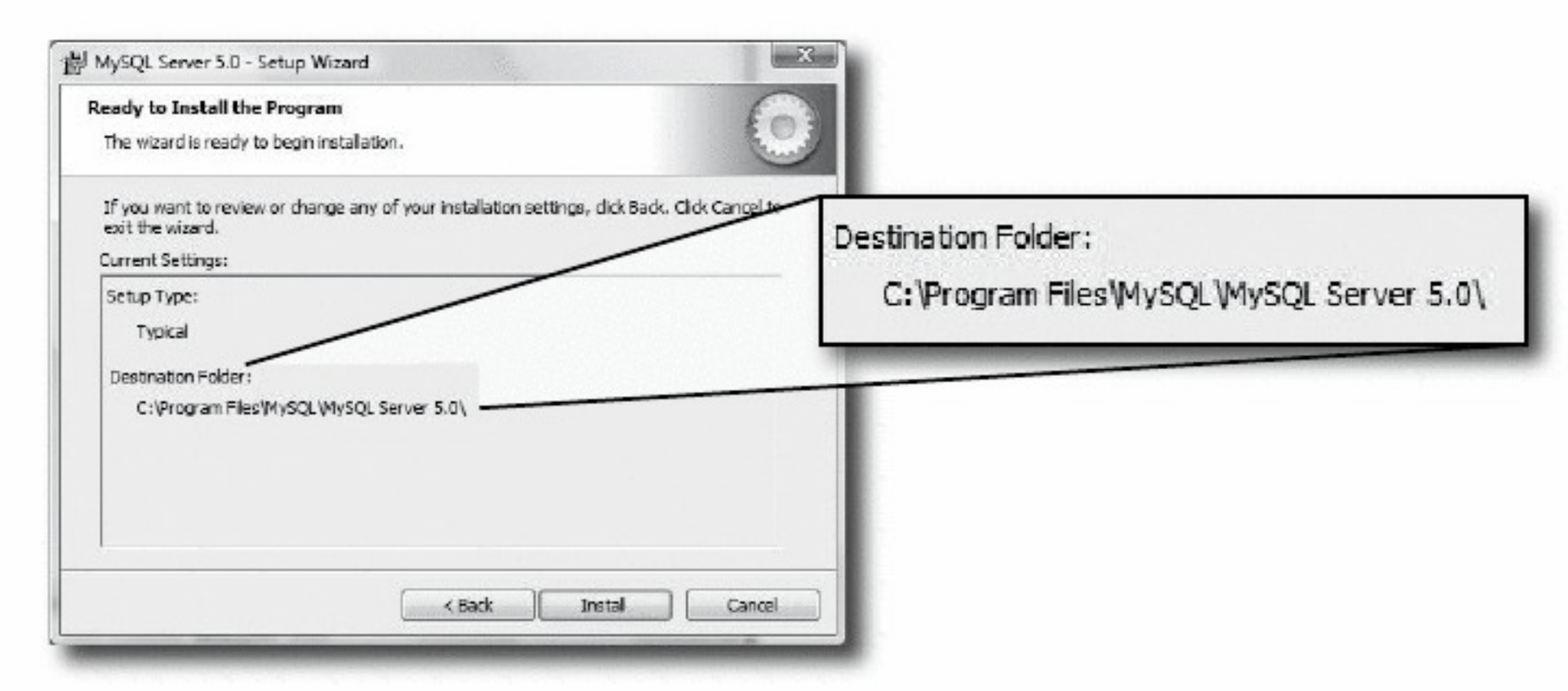
Choisissez un répertoire de destination

6 Vous devez choisir entre Typical, Complete ou Custom. Pour nos objectifs dans ce livre, choisissez **Typical**.

Vous pouvez changer l'endroit de votre ordinateur où MySQL va être installé, mais nous vous recommandons de garder l'option par défaut :

```
C:\Program Files\MySQL\MySQL Server 5.0
```

Cliquez sur le bouton ***Next***.



Cliquez sur Install et c'est fini !

7 Vous verrez la fenêtre « Ready to Install » avec l'indication du répertoire de destination. Si vous êtes satisfait du répertoire de destination, cliquez sur Install. Sinon, cliquez sur ***Back***, ***Changez*** le répertoire, puis revenez ici.

Cliquez sur ***Install***.

Étapes pour installer MySQL sous Mac OS X

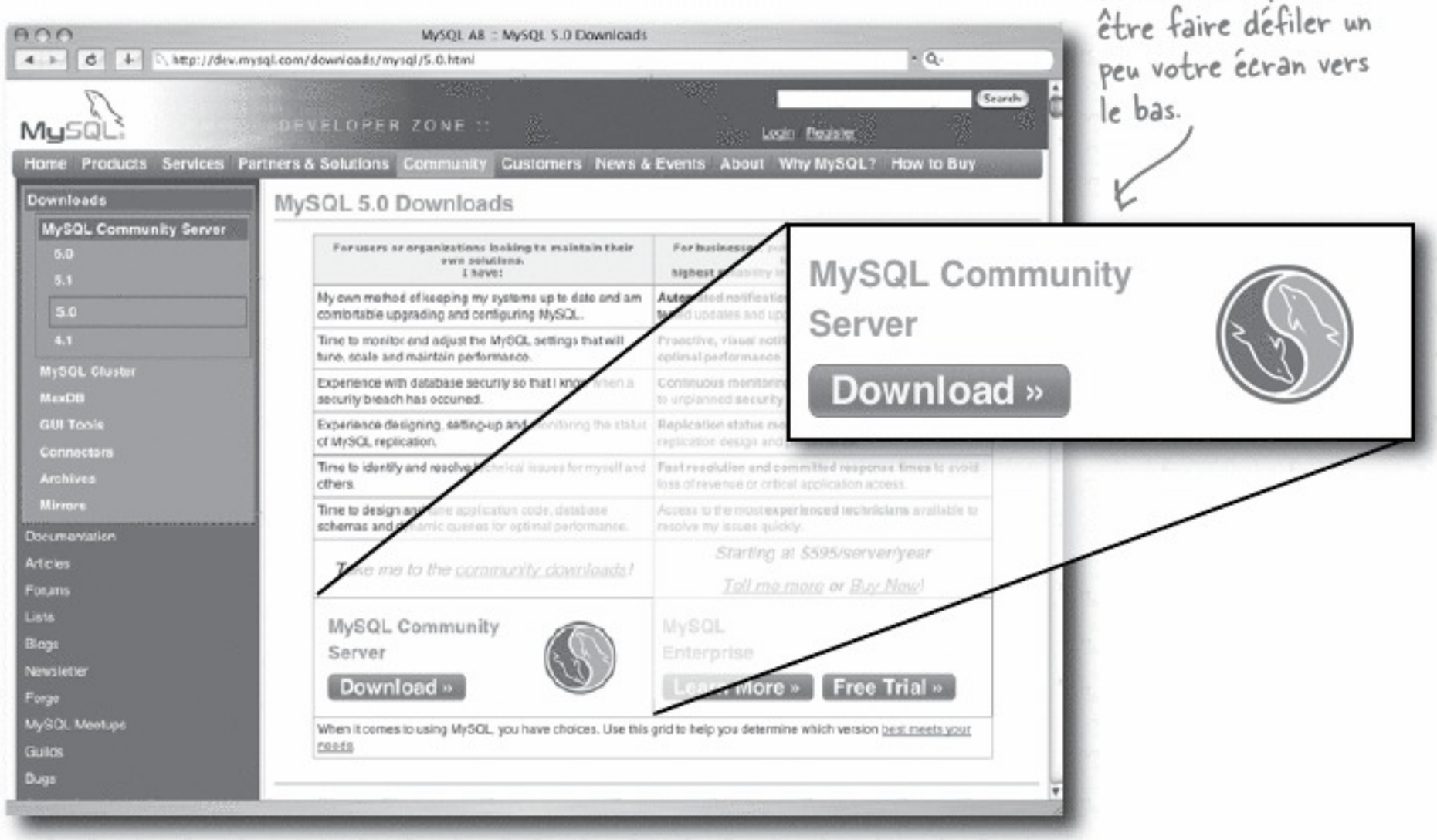
Si vous êtes sous Mac OS X Server, une version de MySQL doit déjà être installée.

Avant de commencer, vérifiez si vous avez déjà une version installée. Allez dans **`Applications/Server/MySQL Manager`** pour y accéder.

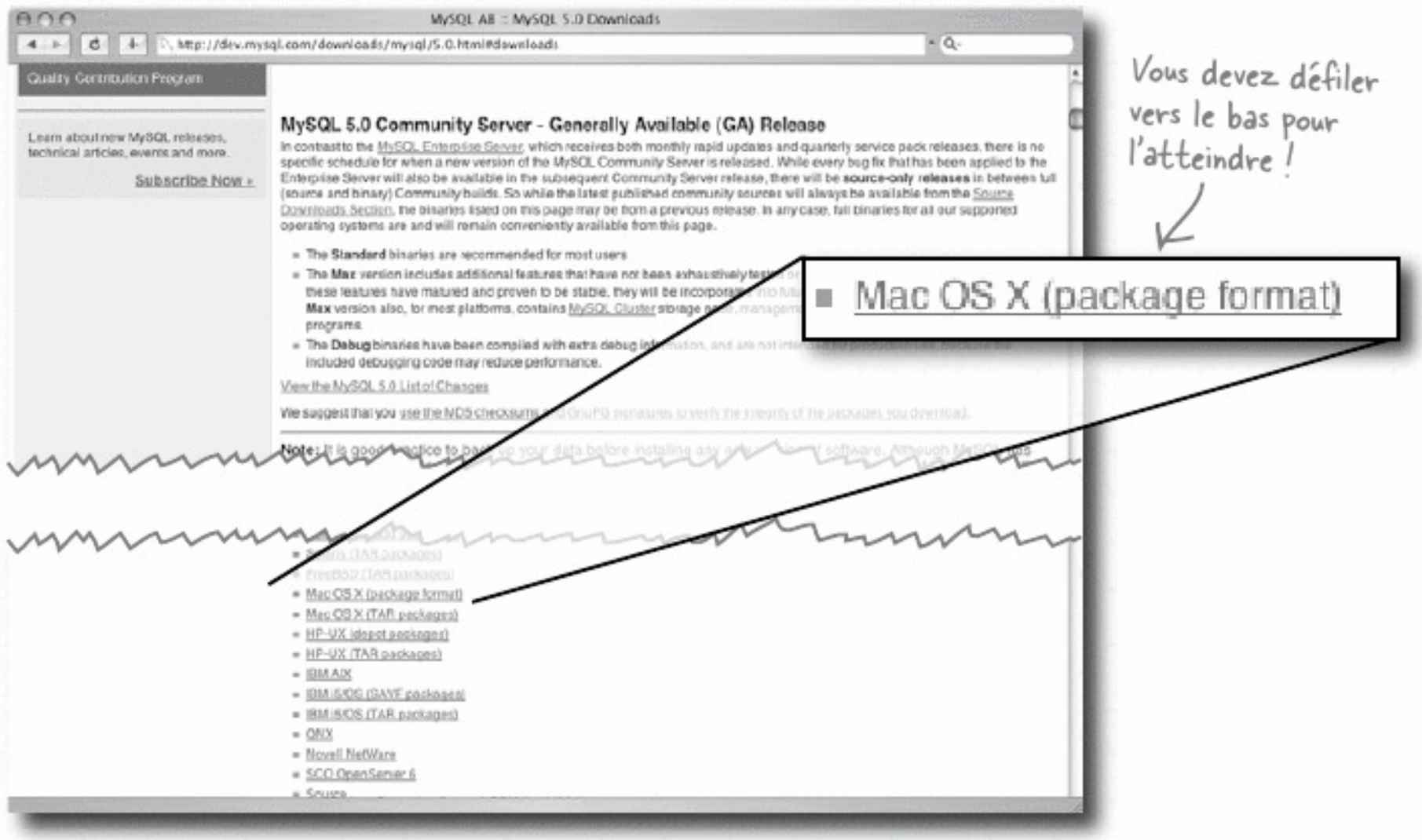
1. Allez sur :

   `http://dev.mysql.com/downloads/mysql/5.0.html`

   et cliquez sur le bouton de téléchargement de MySQL Community Server.



2. Dans la liste, choisissez **Mac OS X (package format)**.



3. Choisissez le package correspondant à votre version de Mac OS X. Cliquez sur **Pick a Mirror**.

4. Vous verrez une liste des serveurs qui ont une copie que vous pouvez télécharger ; choisissez le plus proche de vous.

5. Quand le fichier est téléchargé, double-cliquez dessus pour le lancer. Une fois que vous avez installé MySQL, consultez la documentation en ligne pour trouver comment accéder à votre installation en utilisant le navigateur de requêtes dont nous avons parlé pages 526–527.

Mais si vous êtes pressé, voici un raccourci en utilisant le Terminal.

Vous pouvez ouvrir une fenêtre de Terminal sur votre Mac et taper :

```
shell> cd /usr/local/mysql
shell> sudo ./bin/mysqld_safe
```

(Saisissez votre mot de passe si nécessaire)

(Appuyez sur Contrôle-Z)

```
shell> bg
```

(Appuyez sur **Contrôle-D** ou saisissez **exit** pour sortir)

Tous vos nouveaux outils SQL

Voici tous vos outils SQL réunis au même endroit pour la première fois, pour une unique soirée ! Plus sérieusement, vous êtes au point de rendez-vous de tous les outils SQL que nous avons traités. Prenez un moment pour *examiner la liste et vous sentir **fier*** : **vous les avez tous appris** !

Symboles

= <> < > <= >=

Vous avez de nombreux opérateurs d'égalité ou d'inégalité à votre disposition.

chapitre 2

A

ALTER avec ADD

Vous permet d'ajouter une colonne à votre table et de choisir où la placer.

chapitre 5

ALTER avec CHANGE

Vous permet de changer à la fois le nom et le type de données d'une colonne existante.

chapitre 5

ALTER avec MODIFY

Vous permet de ne changer que le type de données d'une colonne existante.

chapitre 5

ALTER avec DROP

Vous permet d'envoyer un drop à une colonne de votre table pour la supprimer.

chapitre 5

ALTER TABLE

Vous permet de changer le nom de votre table et sa structure toute entière tout en gardant les données qui s'y trouvent.

chapitre 5

AND et OR

Avec AND et OR, vous pouvez combiner vos instructions conditionnelles dans vos clauses WHERE pour plus de précision.

chapitre 2

AUTO_INCREMENT

Si vous l'utilisez dans votre déclaration de colonne, cette colonne recevra automatiquement une valeur entière unique à chaque fois qu'une commande INSERT est exécutée.

chapitre 4

AUTO-JOINTURE

L'AUTO-JOINTURE vous permet d'interroger une table comme s'il y en avait deux, contenant exactement les mêmes informations.

chapitre 10

AVG

Renvoie la valeur moyenne d'une colonne numérique.

chapitre 6

B

BETWEEN

Vous permet de sélectionner des ensembles de valeurs.

chapitre 2

C

CLÉ ÉTRANGÈRE

C'est une colonne de la table qui référence la clé primaire d'une autre table.

chapitre 7

CLÉ ÉTRANGÈRE AUTO-RÉFÉRENCÉE

C'est une clé étrangère dans la même table où elle est clé primaire, mais utilisée dans un autre but.

chapitre 10

CLÉ COMPOSITE

C'est une clé primaire composée de plusieurs colonnes, ce qui crée une valeur de clé unique.

chapitre 7

CLÉ PRIMAIRE

Une colonne ou un ensemble de colonnes qui identifie de façon unique une ligne de données dans une table.

chapitre 4

CHECK OPTION

Utilisez-le quand vous créez une vue modifiable pour forcer tous les insert et les update à remplir une clause WHERE dans la vue.

chapitre 11

CONTRAINTES CHECK

Utilisez-les pour ne permettre qu'à certaines valeurs précises d'être insérées ou modifiées dans une table.

chapitre 11

COUNT

Peut vous dire combien de lignes correspondent à une requête SELECT sans que vous ayez à voir les lignes. COUNT renvoie une seule valeur entière.

chapitre 6

CREATE DATABASE

Utilisez cette instruction pour définir la base de données qui va contenir toutes vos tables.

chapitre 1

CREATE TABLE

Commence à définir votre table mais vous avez aussi besoin de connaître vos NOMS DE COLONNES et TYPES DE DONNÉES. Vous devez les trouver en analysant le genre de données que vous allez entrer dans votre table.

chapitre 1

CREATE TABLE AS

Utilisez cette commande pour créer une table à partir des résultats d'une instruction SELECT.

chapitre 10

CREATE USER

Utilisé par certains SGBDR pour vous permettre de créer un utilisateur et de lui donner un mot de passe.

chapitre 12

CROSS JOIN : JOINTURE CROISÉE

Renvoie toutes les lignes d'une table, croisées avec toutes les lignes d'une seconde table. Connue sous beaucoup d'autres noms y compris JOINTURE CARTÉSIENNE et NON-JOINTURE.

chapitre 8

D

DEFAULT

Vous permet de préciser une valeur par défaut pour une colonne, utilisée si vous ne fournissez pas de valeur pour la colonne quand vous insérez un enregistrement.

chapitre 1

DELETE

C'est l'outil pour supprimer des lignes de données de votre table. Utilisez-le avec une clause WHERE pour cibler précisément les lignes que vous voulez enlever.

chapitre 3

DÉPENDANCE FONCTIONNELLE TRANSITIVE

Cela signifie qu'une colonne non-clé est liée à une autre colonne non-clé.

chapitre 7

DEUXIÈME FORME NORMALE (2FN)

Votre table doit être en 1FN et ne contenir aucune dépendance fonctionnelle partielle pour être en 2FN.

chapitre 7

DISTINCT

Renvoie chaque valeur une seule fois, sans duplicats.

chapitre 6

DONNÉES ATOMIQUES

Les données de vos colonnes sont atomiques si elles sont divisées en morceaux, les plus petits dont vous ayez besoin.

chapitre 4

DONNÉES ATOMIQUES RÈGLE 1 :

Les données atomiques ne peuvent pas avoir plusieurs morceaux du même genre de données dans la même colonne.

chapitre 4

DONNÉES ATOMIQUES RÈGLE 2 :

Les données atomiques ne peuvent pas avoir plusieurs colonnes avec le même genre de données.

chapitre 4

DROP TABLE

Vous permet de supprimer une table si vous faites une erreur. Mais vous devez faire cela avant de commencer à utiliser des instructions INSERT vous permettant d'ajouter des valeurs dans chaque colonne.

chapitre 1

E

ÉQUI-JOINTURE et NON ÉQUI-JOINTURE

Des jointures internes toutes les deux. L'ÉQUI-JOINTURE renvoie des lignes qui sont égales et la NON ÉQUI-JOINTURE renvoie toutes les lignes qui ne sont pas égales.

chapitre 8

EXCEPT

Utilisez ce mot-clé pour ne renvoyer que les valeurs de la première requête ne se trouvant PAS dans la seconde.

chapitre 10

F

Fonctions de chaînes

Vous permettent de modifier des copies du contenu des colonnes de chaînes quand elles sont renvoyées par une requête. Les valeurs originales restent inchangées.

chapitre 5

G

GRANT

Vous permet de contrôler exactement ce que les utilisateurs peuvent faire dans les tables et les colonnes, à partir des privilèges que vous leur donnez.

chapitre 12

GROUP BY

Regroupe des lignes à partir d'une colonne commune.

chapitre 6

I

INNER JOIN : JOINTURE INTERNE

Une jointure qui combine les enregistrements de deux tables en utilisant une condition.

chapitre 8

INTERSECT

Utilisez ce mot-clé pour ne renvoyer que des valeurs se trouvant dans la première ET la seconde requête

chapitre 10

IS NULL

Utilisez ceci pour créer une condition qui traque cette nuisance de valeur NULL.

chapitre 2

J

JOINTURE EXTERNE DROITE

Un RIGHT OUTER JOIN prend toutes les lignes de la table de droite et les fait correspondre à celles de la table de GAUCHE

chapitre 10

JOINTURE EXTERNE GAUCHE

Un LEFT OUTER JOIN prend toutes les lignes de la table de gauche et les fait correspondre à celles de la table de DROITE.

chapitre 10

JOINTURE VIRGULE

La même chose qu'une JOINTURE CROISÉE, sauf qu'on utilise une virgule au lieu des mots-clés CROSS JOIN.

chapitre 8

L

LIKE avec % et _

Utilisez LIKE avec les jokers pour chercher dans des parties de chaînes de caractères.

chapitre 2

LIMIT

Vous permet de préciser exactement le nombre de lignes à renvoyer et avec quelle ligne commencer.

chapitre 6

M

MAX et MIN

Renvoie la plus grande valeur d'une colonne pour MAX et la plus petite pour MIN.

chapitre 6

N

NATURAL JOIN : JOINTURE NATURELLE

Une jointure interne sans la clause « *ON* ». Cela ne fonctionne que si vous joignez deux tables qui ont un même nom de colonne.

chapitre 8

NOT

NOT vous permet de rendre vos résultats négatifs et d'obtenir les valeurs opposées.

chapitre 2

NULL et NOT NULL

Vous avez également besoin de savoir quelles colonnes ne doivent pas accepter de valeurs NULL pour vous aider à trier et fouiller dans vos données. Vous devez définir les colonnes comme NOT NULL quand vous créez votre table.

chapitre 1

O

ORDER BY

Trie vos résultats dans l'ordre alphabetique à partir d'une colonne que vous indiquez.

chapitre 6

P

PREMIÈRE FORME NORMALE (1FN)

Chaque ligne de données doit contenir des valeurs atomiques et avoir un identifiant unique.

chapitre 4

R

RELATION PLUSIEURS-À-PLUSIEURS

Deux tables sont reliées par une table de jonction, permettant à plusieurs lignes de la première de correspondre à plusieurs lignes de la seconde et inversement.

chapitre 7

RELATION UN-À-PLUSIEURS

Une ligne d'une table peut avoir plusieurs lignes correspondantes dans la seconde table, mais la seconde table ne peut avoir qu'une ligne correspondante dans la première.

chapitre 7

RELATION UN-À-UN

Très exactement une ligne d'une table parente est

reliée à une ligne d'une table enfant.

chapitre 7

REQUÊTE EXTERNE

Une requête qui contient une `requête interne` ou `sous-requête`.

chapitre 9

REQUÊTE INTERNE

Une requête dans une autre requête. Également appelée une `sous-requête`.

chapitre 9

REVOKE

Utilisez cette instruction pour retirer des privilèges à un utilisateur.

chapitre 12

RÔLE

Un rôle est un groupe de privilèges. Les rôles vous permettent de regrouper des privilèges spécifiques et de les assigner à plus d'un utilisateur.

chapitre 12

S

**S'ÉCHAPPER AVEC ' ET **

Échappez aux apostrophes de vos données texte en plaçant un guillemet simple supplémentaire ou un antislash devant lui.

chapitre 2

SCHÉMA

Une description des données de votre base de données ainsi que de tout autre objet qui s'y rapporte et de la façon dont ils sont tous liés entre eux.

chapitre 7

**SELECT *

Utilisez ceci pour sélectionner toutes les colonnes d'une table.

chapitre 2

SET

Ce mot-clé fait partie d'une instruction `UPDATE` et s'utilise pour modifier la valeur d'une colonne existante.

chapitre 3

SHOW CREATE TABLE

Utilisez cette commande pour voir la syntaxe correcte pour créer une table existante.

chapitre 4

SOUS-REQUÊTE

Une requête enveloppée dans une autre requête. Également appelée une `requête interne`.

chapitre 9

SOUS-REQUÊTE CORRÉLÉE

Une `sous-requête` qui s'appuie sur des valeurs renvoyées par la `requête externe`.

chapitre 9

SOUS-REQUÊTE NON-CORRÉLÉE

Une `sous-requête` qui est indépendante et ne fait aucune référence à la `requête externe`.

chapitre 9

START TRANSACTION

Est utilisé pour dire au SGBDR de commencer une

transaction. Rien n'est permanent jusqu'à ce que COMMIT soit envoyé. La transaction va continuer jusqu'à ce qu'une commande COMMIT ou ROLLBACK soit entrée, cette dernière renvoyant la base de données à son état antérieur à START TRANSACTION.

chapitre 11

SUM

Calcule le total d'une colonne de valeurs numériques.

chapitre 6

T

TRANSACTIONS

C'est un groupe de requêtes qui doivent être exécutées ensemble, comme une unité. Si elles ne peuvent pas toutes s'exécuter sans interruption, alors aucune ne l'est.

chapitre 11

TROISIÈME FORME NORMALE (3FN)

Votre table doit être en 2FN et ne pas avoir de dépendances transitives.

chapitre 7

U

UNION et UNION ALL

Une UNION combine les résultats de deux ou plus de requêtes dans une table, à partir de ce que vous précisez dans la liste des colonnes du SELECT. UNION cache les valeurs dupliquées, UNION ALL les inclut.

chapitre 10

UPDATE

Cette instruction met à jour une ou plusieurs colonnes avec une nouvelle valeur. Elle utilise également une clause WHERE.

chapitre 3

USE DATABASE

Vous permet d'entrer dans la base de données pour définir toutes vos tables.

chapitre 1

V

VUES

Utilisez une vue pour traiter les résultats d'une requête comme une table. Génial pour transformer des requêtes complexes en requêtes simples.

chapitre 11

VUES MODIFIABLES

Ce sont les vues qui vous permettent de modifier les données dans les tables sous-jacentes. Ces vues doivent contenir toutes les lignes NOT NULL de la ou des tables de base.

chapitre 11

VUES NON-MODIFIABLES

Des vues qui ne peuvent pas être utilisées pour insérer ou supprimer des données dans la table de base.

chapitre 11

W

WITH ADMIN OPTION

Permet à quelqu'un ayant un rôle d'assigner ce rôle à quelqu'un d'autre.

chapitre 12

WITH GRANT OPTION

Permet aux utilisateurs de donner à d'autres utilisateurs les privilèges qu'ils ont.

chapitre 12

Index

Symboles

\ (antislash)
* (astérique)
    SELECT, instruction 58, 77
<> (différent de) 86
= (égal) 86
<= (inférieur ou égal à) 87
> (plus grand que) 85
% (signe pourcentage) 101
>= (supérieur ou égal à) 87
_ (tiret bas) 102

Nombres

1 FN (première forme normale) 320
2FN (deuxième forme normale) 329, 330, 554
3FN (troisième forme normale) 336

A

ABS( ), fonction 537
accès distant 497
ACID
    atomicité 483
    cohérence 483
    durabilité 483
    isolation 483
ACOS( ), fonction 537
ADD, mot-clé 232, 552
ADD CONSTRAINT CHECK 461
ADD INDEX 539
ADD PRIMARY KEY, instruction 210
AFTER, mot-clé 200, 202
alias 354–356
    colonnes 355
    Révélations sur les alias de tables et de colonnes 376
    tables 356
ALL, mot-clé 530
ALTER, instruction 191
    clé primaire 193
ALTER TABLE, instruction 197–234, 232, 552
    ADD, mot-clé 232, 552
    ADD PRIMARY KEY, instruction 210
    AFTER, mot-clé 200, 202
    AUTO_INCREMENT, mot-clé 200, 210, 220
    BEFORE, mot-clé 201, 202
    CHANGE, mot-clé 232, 552
    CHANGE COLUMN, instruction 210
        changer des colonnes multiples 211
    changer
        des colonnes 208–209
        des types de données 203
        une table 200
    clés étrangères 306
    DROP, mot-clé 232, 248, 552
    DROP COLUMN, instruction 216
    DROP PRIMARY KEY, instruction 220
    FIFTH, mot-clé 202
    FIRST, mot-clé 200, 202
    LAST, mot-clé 201, 202
    modifier précisément la table 200
    MODIFY, mot-clé 212, 232, 552
    NOT NULL, mot-clé 200
    RENAME, instruction 207
    renommer les tables 205

ALTER TABLE, instruction (suite)
SECOND, mot-clé 201, 202
SELECT, instruction 221
THIRD, mot-clé 201
UPDATE, instruction 221
anatomie de
base de données 10
instruction 34
une requête dans une requête 388
AND, opérateur, utiliser plusieurs 284
AND, requêtes 80, 95
utiliser NOT 111
versus OR 96
antislash (échapper les caractères) 67, 68
ANY, mot-clé 531
apostrophes 18
données ayant des 68
échapper avec un antislash 68
AS, mot-clé 353–354
ASC, mot-clé 261
ascendant, ordre 260–261
ASIN( ), fonction 537
astérisque 58
ATAN( ), fonction 537
atomiques, données (voir données atomiques)
auto-jointure 435, 452, 552
comme sous-requête 449
AUTO_INCREMENT 188, 194, 200, 220, 552
ALTER, instruction 210
clés primaires 192
AVG, fonction 267, 278
vues 476

B

base de données
à retenir 27
cylindre 8
données reliées entre elles 12
nommer 18
quotidiennes 9
représentation 8
(voir aussi tables)
BDB 486
BEFORE, mot-clé 201, 202
BETWEEN, mot-clé 106
NOT 111
vues 476
BIGINT, type de données 532
BLOB, type de données 24, 25
interroger 64
Boîte à outils SQL
1FN (première forme normale) 340
2FN (deuxième forme normale) 340, 554
ALTER avec
ADD 232
CHANGE 232
DROP 232
MODIFY 232
ALTER TABLE, instruction 232, 552
AND et OR 116, 552
auto-jointure 452, 552
AUTO_INCREMENT, mot-clé 194, 552
AVG, fonction 278, 552
BETWEEN 116, 552
CHECK, contraintes 490, 553
CHECK OPTION 490, 553
clés 194
composites 340, 553
étrangères 340, 555
étrangères auto-référencées 452, 553
primaires 553
COUNT, fonction 278, 553
CREATE DATABASE, commande 50, 553
CREATE TABLE, commande 50, 553

Boîte à outils SQL (suite)
CREATE TABLE AS 452, 553
CREATE USER, instruction 522, 553
CROSS JOIN 377, 555
DEFAULT, mot-clé 50, 554
DELETE, instruction 158, 554
dépendance fonctionnelle transitive 340
DESC, instruction 50
DISTINCT, mot-clé 278, 554
données atomiques 194, 554
DROP TABLE, commande 50, 554
échapper les apostrophes 116, 557
équi-jointure et non-équi-jointure 377, 554
EXCEPT 452, 554
fonctions de chaînes 232
GRANT, instruction 522, 555
GROUP BY, instruction 278, 555
INNER JOIN 377, 555
INSERT, instruction 50
INTERSECT 452, 555
IS NULL 116, 555
jointure externe
droite 452
gauche 452
jointure virgule 377
LEFT OUTER JOIN 452, 555
LIKE et jokers 116, 555
LIMIT, instruction 278, 555
MAX et MIN, fonctions 278, 556
NATURAL JOIN 377, 556
NOT 116, 556
NULL et NOT NULL 50, 556
ORDER BY, instruction 278, 556
relations
plusieurs-à-plusieurs 340, 556
un-à-plusieurs 340, 556
un-à-un 340, 557
requête
externe 414, 557
interne 414, 557
REVOKE, instruction 522, 557
RIGHT OUTER JOIN 452, 555
rôles 522, 557
schéma 340, 557
SELECT 116, 557
SET, mot-clé 158, 557
SHOW CREATE TABLE, instruction 194
sous-requête 414
corrélée 414
non-corrélée 414
START TRANSACTION 490, 557
SUM, fonction 278, 558
transactions 490, 558
UNION et UNION ALL 452, 558
UPDATE, instruction 158, 558
USE DATABASE, commande 50, 558
vues 490, 558
modifiables 490, 558
non modifiables 490, 558
WITH ADMIN OPTION 522, 559
WITH GRANT OPTION 522, 559
booléen, type de données 532

C

caractères invisibles 77
caractères réservés
* (astérisque) 58
% (pourcentage) 101
apostrophe 18
caractères spéciaux 67
% (pourcentage) 102
_ (tiret bas) 102
backslash 67
langage SQL 528
CASCADE, mot-clé 506, 509
revoking roles 515
révoquer les rôles 514

CASE, expression 241–243
ELSE 246
CAST( ), fonction 535
catégories
de données 7, 14
sélectionner par 250
trier par 253, 257
CEIL( ), fonction 537
cellule 388
CHANGE, mot-clé 220
CHANGE COLUMN, instruction 210
changer plusieurs colonnes 211
CHAR, type de données 24, 25, 64, 66
interroger 64
CHECK, contraintes 460, 461, 462, 477, 490, 553
CHECK OPTION 475, 490, 553
clés
composites (voir clés composites)
étrangères (voir clés étrangères)
naturelles 180
parentes 304
primaires (voir clés primaires)
clés composites 321, 340, 553
clés étrangères 304, 340, 555
auto-référencées 432, 452, 553
contraintes 305
créer 304–305
NULL 304
utilité 306
clés naturelles 180
clés primaires 553
ADD PRIMARY KEY, instruction 210
ajouter à des tables 192
AUTO_INCREMENT, mot-clé 192
clés composites 322
CREATE TABLE, instruction 182–187, 186
DROP PRIMARY KEY, instruction 220
CocoaMySQL 527
colonnes 10, 13
ajouter 31
alias 355
ALTER, instruction 208–209
changer les types de données 203
CHANGE, mot-clé (voir CHANGE COLUMN, instruction)
changer l'ordre 41, 220
CHECK, contraintes (voir CHECK, contraintes)
fonctionnellement dépendantes (voir dépendance fonctionnelle)
index 539
MODIFY, mot-clé 212
na pas en afficher certaines 41
ne garder que celles nécessaires 215
nombre de lignes, trouver 269
nommer 18
avec des mots-clés 187
NULL 45
omettre des 41
omettre les noms de 41
préciser des catégories 239
réordonner 220
sans valeurs 42
sélectionner 73
des colonnes précises 71
toutes 58
sous-requêtes
utilisées comme colonnes de SELECT 397
valeurs, moyenne des 268
valeurs d'une colonne, trouver
la plus grande 268
la plus petite 268
combinaison unique 321
commandes
majuscules 18
point-virgule 18

COMMIT 484, 489
comptes, problèmes de partage 510
comptes utilisateur 536
  ajouter 498
  permissions, accorder 499
  rôles 513
  (voir aussi root)
conseils xxxii
contraintes
  des clés étrangères 304
  (voir aussi CHECK, contraintes)
corrélation, noms de 356
correspondances multiples 425
COS( ), fonction 537
COT( ), fonction 537
COUNT, fonction 269, 278, 553
  DISTINCT, mot-clé 271
  vues 476
CREATE DATABASE, commande 17, 50, 553
CREATE TABLE, instruction 19, 20, 50, 553
  clés primaires 182–187, 186
  peupler avec des valeurs distinctes triées par ordre 352
  SHOW 184
  utiliser après la suppression d'une table 32
CREATE TABLE AS 452, 553
  UNION 441
CREATE USER, instruction 518–519, 522, 553
  GRANT, instruction, avec 519
CREATE VIEW, instruction 465
créer une table 19

D

DATE, type de données 23, 24, 64, 66, 533
DATE_FORMAT( ), fonction 533
DATETIME, type de données 24, 533
DB2 xxxii
DEC, type de données 24, 25, 64
DEFAULT, mot-clé 48, 50
DELETE, instruction 119, 129, 131, 158, 554
  CASE, expression 246
  fonctions de chaînes 230
  INSERT, instruction 135
  règles 132
  remplacer par UPDATE 148
  sous-requêtes 391, 392
  utiliser prudemment 140
  utiliser SELECT en premier 143–145
  WHERE, clause 141
dépendance fonctionnelle 323–329
  2FN 331
  éviter 329
  notations abrégées 324
  partielle 325
  transitive 326
DESC, commande 28
DESC, instruction 50
DESC, mot-clé 27–28
  NOT NULL, valeur 47
  ORDER BY, instruction 260
DESCRIBE, instruction 184, 204
deuxième forme normale (2 FN) 329, 330
diagrammes 8
  relier des tables 300
  séparer la conception et lesdonnées 297
DISTINCT, mot-clé 270–271, 278, 554
données
  afficher dans des tables 7
  ajouter à une table 34
  atomiques 167–170
  catégories de 7, 14
  définir 2
  disposer dans une table 7

commandes (suite)
dupliquées 162
lier 12
mettre à jour 146
(voir aussi UPDATE)
organiser 250
récupérer 53–118
saisie sans contraintes 458
séparer de la conception de la table 297
supprimer 144
texte, opérateurs de comparaison 91
trier 257, 259
(voir aussi ORDER BY, instruction)
types de 24
(voir types de données)
données, types de
changer 203
données atomiques 167–170
déplacer des colonnes non atomiques dans une nouvelle table 294
règles 172
données numériques 88
opérateurs de comparaison 88
DROP, instruction
supprimer des rôles 512
DROP, mot-clé 232, 248, 552
DROP COLUMN, instruction 216, 220
DROP PRIMARY KEY, instruction 220
DROP ROLE, instruction 512
DROP TABLE, commande 32, 50, 554
DROP VIEW, instruction 477

E

enregistrements
en double 128
le dernier 128
le plus récent 127, 128
modifier 144
supprimer 131
(voir aussi DELETE, instruction)
équi-jointure 364
description 372
requêtes 366
étoile, caractère 58
EXCEPT 442, 452, 554
EXISTS 410
EXP( ), fonction 537

F

Face à face
Etes-vous interne ou externe 390
Jointure ou Sous-requête, laquelle est la meilleure 445–446, 446–447
fenêtre de ligne de commande 17
FIFTH, mot-clé 202
FIRST, mot-clé 200, 202
FLOOR( ), fonction 537
fonctions numériques 537–538
FORMAT( ), fonction 537
FROM, clause 467
sous-requêtes 391, 392
FULL OUTER JOIN 427

G

GRANT, instruction 499–500, 509, 512, 522, 555
CREATE USER, instruction, avec 519
variations 503
GROUP BY, instruction 278, 384, 555
AVG, fonction 267
MAX, fonction 268
MIN, fonction 268
SUM, fonction 266
groupes, privilèges 511

guillemets
doubles 67
simples 64
antislash 67
caractères spéciaux 67
conseils 77
dépareillés 66
supplémentaires 68

H

HAVING, clause
sous-requêtes 391, 392
vues 476

I

IN, mot-clé 109
vues 476
IN, opérateur 385–387, 392
sous-requêtes non corrélées 403
index 539
INNER JOIN 377, 555
définition 363
InnoDB 486
INSERT, instruction 34, 34–42, 50, 65
antislash 68
apostrophes
antislash 68
CASE, expression 246
colonnes
changer l'ordre 41
ne pas en afficher certaines 41
omettre certaines 41
omettre des noms de 41
sans valeur 42
DELETE, instruction 135
guillemets dépareillés 66
sous-requêtes 392
variations 41
vues 476
instruction
anatomie 34
INT, type de données 24, 25, 64, 532
intégrité référentielle 305
INTERSECT 442, 452, 555
IS NULL 99

J

jointure cartésienne 357–359
jointure externe
correspondances multiples 425
jointure interne 364, 377
équi-jointure (voir équi-jointure)
NATURAL JOIN (voir NATURAL JOIN)
non équi-jointure (voir non équi-jointure)
jointures
auto-jointure 435, 452, 552
comme sous-requête 449
cartésiennes (voir jointure cartésienne)
comparées aux sous-requêtes 443–446
croisées 377, 555
équi-jointure (voir équi-jointure)
joindre une table à elle-même 433–435
NATURAL JOIN (voir NATURAL JOIN)
naturelles (voir NATURAL JOIN) 368
non équi-jointure (voir non équi-jointure)
virgule 555
jokers
% (signe pourcentage) 102
tiret bas 102
jonction, table de 315

L

LAST, mot-clé 201, 202
LEFT(), fonction 224
LEFT OUTER JOIN 419–426, 427, 452, 555
    NULL, valeur 421
    plusieurs correspondance 425
LENGTH 349
liens, imposer un 308
lignes 10, 13
    nombre dans une colonnes, trouver 269
    ordre 127
    supprimer, faire attention 140
LIKE, mot-clé 101
    jokers 102
    NOT 111
    requêtes 163
LIMIT, instruction 274–275, 278
limiter le nombre de résultats 274–275
LN( ), fonction 537
localhost 497
LOG( ), fonction 537

M

majuscules 18
maths, opérateurs 156
MAX, fonction 268, 278, 394, 556
Microsoft Word 77
MIN, fonction 268, 278, 556
    NULL 273
MOD( ), fonction 538
MODIFY, mot-clé 212, 232, 552
    réordonner des colonnes 220
motifs, rechercher 223
mots-clés, comme nom de colonne 187
mots de passe 497
    utilisateur root 518
mots réservés 529
moyenne des valeurs d'une colonne 267
MS SQL Server xxxii
multi-tables, conception 281–342
    colonnes
        à un seul centre d'intérêt 288–289
        avec plusieurs centres d'intérêt 290–292
        déplacer des colonnes non atomiques dans une nouvelle table 294
        fonctionnellement dépendantes (voir dépendance fonctionnelle) 323
    créer une nouvelle table 293–294
    passer d'une table à deux 298–299
    relations entre tables 309
    relier les tables 296
        dans un diagramme 300
        (voir aussi clés étrangères)
    schéma (voir schéma)
    séparer la conception et les données 297
        table de jonction 315
MySQL xxxii
    Administrator 526
    CHECK, contraintes 460, 462
        CHECK OPTION 475
    Community Server 544
    installation
        guide de dépannage 544
        Mac OS X 548
        Windows 545
    outils à interface graphique 526
    PHP et 540–541
    Query Browser 526, 544
    rôles 511
    transactions 486
    vues 476

N

NATURAL JOIN 368, 377, 384–385, 395, 556
  description 372
  IN, opérateur 385–387
  sous-requêtes 398
Navicat 527
noms de corrélation 356
noms des bases, tables et colonnes 18
non équi-jointure 367, 554
  description 372
  requêtes 370
normalisation
  3NF (troisième forme normale) 336
  règles 175
Notepad 31
NOT EXISTS
  sous-requêtes corrélées 409
NOT IN, mot-clé 110
  sous-requête non-corrélée 403
  vues 476
NOT NULL, mot-clé 200
  vues 476
NOT NULL, valeurs 45, 46, 50, 556
  DESC, commande 47
NULL 44, 50, 556
  clés
    étrangères 304
    primaires 179
  contrôler 45
  éviter 458
  LEFT OUTER JOIN 421
  MIN, fonction 273
  NOT (voir NOT NULL, mot-clé)
  sélection de table 99
  trouver dans une table 98
  utiliser NOT 111
  WHERE, clause
    DELETE, instruction 134
  (voir aussi NOT NULL, valeurs)
NULL, valeur 421
numériques, champs 67, 77
numériques, types 25

O

opérateurs aggrégés et vues 476
opérateurs de comparaison 528
  <= (moins grand que ou égal) 87
  <> (différent de) 86
  = (égal) 86
  > (plus grand que) 85
  >= (supérieur ou égal à) 87
  trouver des données numériques 88
opérateurs mathématiques 156, 528
OR, requête 93, 95
  utiliser NOT 111
  versus AND 96
Oracle xxxii
ORDER BY, instruction 253–254, 278, 384, 556
  ASC, mot-clé 261
  ASCENDING, mot-clé 260–261
  DESC, mot-clé 260
  ordre
    ascendant 261
    descendant 261
ordre
  des lignes 127
  des résultats 260

OUTER JOIN 428
FULL 427
LEFT 419–426, 427, 452, 555
NULL, valeur 421
plusieurs correspondances 425
plusieurs correspondances 425
RIGHT 426–427, 452, 555
schéma 429–431
outils d'interface graphique
CocoaMySQL 527
MySQL 526
Navicat 527
phpMyAdmin 527
SQLyog 527
outils de transaction SQL 484
Ouvrez les yeux !
CHECK, contraintes 460
DROP COLUMN, instruction 216
INTERSECT et EXCEPT 442
MySQL, rôles 511
tables temporaires 534
type de données
choisir 59
modifier 211
utilisateurs, gérer 498

P

parenthèses 19
sous-requêtes 392
patterns
fonctions de chaînes 224
plusieurs-à-plusieurs 312, 316
un-à-plusieurs 311
un-à-un 309
utilisation 310
permissions 499
pour plusieurs utilisateurs 509
PHP 73
guillemets doubles 67
MySQL et 540–541
phpMyAdmin 527
PI( ), fonction 538
plusieurs-à-plusieurs, tables 312, 316, 340
point-virgule 19
sous-requêtes 392
Points d'impact 27, 50, 77, 220
PostgreSQL xxxii
POWER( ), fonction 538
première forme normale (1FN) 319
principes d'apprentissage xxviii

R

RADIANS( ), fonction 538
RAND( ), fonction 538
RDBMS
GUI tools 526
récupérer des données 53–118
relations entre tables 164, 309
(voir aussi patterns)
RENAME, instruction 205, 207, 220
requêtes 8
à partir d'un navigateur web 77
combiner 80
compliquées 284
externes 386–387
internes 386–387, 414, 557
jointures 370
LIKE, instruction 163
longueur 163
vues (voir vues)
(voir aussi sous-requêtes)
RESTRICT, mot-clé 506
révoquer les rôles 515

résultats
limiter le nombre 274–275
Révélations sur SQL
Choisir la meilleure requête 400
Confessions d'un NULL 44
REVOKE, instruction 504, 522, 557
CASCADE, mot-clé 506, 509
rôles 514, 515
RESTRICT, mot-clé 506
roles 515
REVOKE GRANT OPTION 504–505
RIGHT(), fonction 224
RIGHT OUTER JOIN 426–427, 452, 555
rôles 511, 522, 557
dans MySQL 511
REVOKE, instruction
CASCADE keyword 514, 515
RESTRICT keyword 515
supprimer 512, 513
utilisateurs en ayant plusieurs 513
utiliser 512
WITH ADMIN OPTION 514
ROLLBACK 484, 489
root, compte 497
ROUND( ), fonction 538

S

schéma 296, 340, 557
passer d'une table à deux 298–299
relier les tables 300
SECOND, mot-clé 201, 202
sécurité
compte root 497
comptes utilisateur
accorder des permissions 499
ajouter 498
GRANT, instruction (voir GRANT, instruction)
groupes, privilèges (voir rôles)
mot de passe 497
permissions (voir permissions)
REVOKE, instruction (voir REVOKE, instruction)
SELECT, instruction 43, 53–118, 54
* (astérisque) 58, 77
CASE, expression 246
fonctions de chaînes 230
GROUP BY, instruction 266
AVG function 267
limiter les résultats 77
multiple 78–79
ORDER BY, instruction 253
organiser les données 250
sélectionner
des colonnes précises 71
des données précises 71
séparer les valeurs avec des sous-chaînes 351
sous-requêtes 392
expression d'une colonne 397
SUM, fonction 265–266
tris
BETWEEN, mot-clé 106
UNION 438
utiliser avant DELETE 143–145
WHERE, clause (voir WHERE, clause)
SELECT CURRENT_USER 536
sélectionner
des données 71
sous-chaînes 224
SET, mot-clé 147, 158, 557
CASE, expression 242–243
multiple 147
préciser des catégories 239
UPDATE, instruction 147, 154

SGBDR xxxii, 17
outils d'interface graphique 526
SHOW COLUMNS, instruction 187
SHOW CREATE DATABASE, instruction 187
SHOW CREATE TABLE, instruction 184, 194
guillemet oblique et 187
SHOW INDEX, instruction 187
SHOW WARNINGS, instruction 187
SIGN( ), fonction 538
SIN( ), fonction 538
SMALLINT data type 532
SOME, mot-clé 531
sous-chaînes
séparer des valeurs avec 348
sous-requêtes
alias de colonnes 393
anatomie 388
auto-jointure comme 449
au travail 389
comparées aux jointures 443–446
corrélées
EXISTS 410
NOT EXISTS 409–410
dans une autre 412
expression d'une colonne 397
FROM, clause 391, 392
HAVING, clause 391, 392
NATURAL JOIN 398
parenthèses 392
point-virgule 392
requêtes
externes 386–387
internes 386–387
transformer en jointures 444
valeur scalaire 388
sous-requêtes corrélées
EXISTS 410
NOT EXISTS 409–410
SQL, langage
caractères spéciaux 528
mots réservés 529
SQLyog 527
SQRT( ), fonction 538
Standard SQL xxxii
START TRANSACTION 484, 489, 490, 557
SUBSTR, fonction 349
SUBSTRING_INDEX(), fonction 224, 290, 341, 348
SUM, fonction 265–266, 278, 558
vues 476
supprimer des tables 32

T

tables 10
accorder des privilèges pour toutes 513
ALTER TABLE, instruction 221–223
chercher dans 56
conception intelligente 167
normalisation 177
créer 19, 31, 164
avec des clés étrangères 307
créer rapidement
SHOW CREATE TABLE 185
définir 8
diagramme 297
données atomiques 170
joindre plusieurs 375
lignes et colonnes 10
modifier 499
sans endommager les données 31
multiples
nommer 18

tables (suite)
noms de corrélation 356
permissions 499
points-clés 27
relationnelles 164
relations entre tables 309
relier les données 12
renommer 205
supprimer 32
temporaires 534
virtuelles 468
(voir aussi multi-tables, conception)
TAN( ), fonction 538
texte, champ de 67
TextEdit 31, 77
THIRD, mot-clé 201
TIME, type de données 24, 64, 533
interroger 64
TIMESTAMP, type de données 24, 533
tiret bas 18
transactions 478–489, 490, 558
ACID
atomicité 483
cohérence 483
durabilité 483
isolation 483
COMMIT 484, 489
définition 482
gérer 484
MySQL 486
outils de 484
ROLLBACK 484, 489
START TRANSACTION 484, 489
tris
BETWEEN, mot-clé 106
troisième forme normale (3FN) 336
TRUNCATE( ), fonction 538
types de données 24, 532–533
BIGINT 532
BLOB (voir BLOB, type de données)
boolean 532
CAST( ), fonction, et 535
CHAR (voir CHAR, type de données)
DATETIME (voir DATETIME, type de données)
DEC (voir DEC, type de données)
INT (voir INT, type de données)
interroger 64
SMALLINT 532
TIME (voir TIME, type de données)
TIMESTAMP (voir TIMESTAMP, type de données)
VARCHAR (voir VARCHAR)
WHERE, clause 64

U

UNION 437–441, 452, 558
limites 438
règles 438
en action 439
UNION ALL 440, 452, 558
UPDATE, instruction 119, 146, 158, 227–230, 558
CASE, expression 246–248
en action 149
fonctions de chaînes 230
mettre à jour 147
prix 154
ordres des instructions 240
plusieurs enregistrements 156
règles 147
sans clause WHERE 147
SET, instruction 239
sous-requêtes 391, 392
USE, commande 18
USE DATABASE, commande 50, 558

V

valeurs agrégées 471
dupliquées 345
peupler des tables 352
scalaires 388
séparer avec des spus-chaînes 348
valeurs numériques
trouver 83
VARCHAR, type de données 19, 23, 24, 25, 64, 66
vues 464–468, 472, 490, 558
avantages 468
AVG 476
BETWEEN 476
CHECK OPTION 475
COUNT 476
créer 465
DROP VIEW, instruction 477
HAVING 476
IN 476
les voir 466, 477
modifiables 476, 490, 558
MySQL 476
non modifiables 476, 490, 558
NOT IN 476
NOT NULL 476
opérateurs aggrégés et 476
rôles 467
SUM 476
table virtuelle 468
valeurs aggrégées 471
WHERE, clause 476

W

WHEN
ELSE 246
WHERE, clause 57, 63, 143
AND et OR, requêtes 95
différences 96
BETWEEN, mot-clé
NOT 111
DELETE, instruction 129, 132
NULL 134
utiliser SELECT en premier 143–145
IN, mot-clé 109
LIKE, mot-clé 101
NOT 111
NOT IN, mot-clé 110
SELECT versus DELETE 132
tris
BETWEEN, mot-clé 106
types de données 64
UPDATE , instruction 146, 154
vues 476
WHERE, condition 131
WITH ADMIN OPTION 514, 522, 559
WITH GRANT OPTION 501–504, 502–505, 508, 522, 559

Maintenant que vous avez appliqué l'approche Tête la première à SQL, pourquoi ne pas l'appliquer au reste de votre existence ?

Venez nous rejoindre sur le site web des Tête la Première, notre repaire virtuel où vous trouverez toutes sortes de ressources, notamment des forums, des exemples de code, et bien d'autres choses encore.

Mais vous ne resterez pas spectateur. Nous vous encourageons aussi à participer en vous joignant aux discussions et aux échanges d'idées.

Vos avantages ?

- Lisez les dernières nouvelles sur ce qui se passe dans le monde Tête la première.
- Participez à nos technologies et à nos ouvrages à paraître.
- Apprenez comment aborder ces sujets si spéciaux (dites-le trois fois très vite) en aussi peu de temps que possible.
- Découvrez comment les livres Tête la première sont créés.
- Rencontrez ceux qui font que tout baigne dans l'huile...
- Découvrez aussi le site américain à http://www.headfirstlabs.com.

http://www.tetelapremiere.fr