Introduction aux bases de données

Notes de cours

Johnny Piette

Table des matières

I. Introduction

1. Mise en situation

Vous avez fait un super programme en Python qui va faire la moyenne des points obtenus pour le cours de Python. Le résultat est stocké en mémoire. Vous avez encodé les résultats de 4 classes composées de 30 élèves chacune. C'est cool. Vous avez le résultat désiré : la moyenne de la classe en python de chaque classe et la moyenne totale de toutes classes confondues. Bien joué, vous assurez ! 😊

Vous éteignez votre ordinateur. Vous allez en terrasse car elles ont réouvert le 8 mai. Et boum, le lendemain, mal de crâne et le trou noir de la journée précédente. Votre directeur vous téléphone et vous demande la moyenne/classe et la moyenne générale. Mais impossible de vous en souvenir… Aie aie aie cette journée commence très mal…

2. Brainstorming : Qu'auriez-vous pu faire pour garder les résultats ?

A partir de l'exemple précédent que peut-on retirer comme conclusion ?

3. Approche intuitive des SGBD

De la faiblesse de notre programme stocké en mémoire. On constate que l'on aurait pu stocker notre résultat dans un fichier. Et le relire par après. Ce qui est déjà une très grande évolution par rapport à un stockage en mémoire vive.

Cependant, notre directeur a dû nous téléphoner pour avoir le résultat. Il est déjà dommage d'avoir eu besoin de transmettre ce fichier stocké sur notre ordinateur.

Dans une infrastructure de type entreprise, ça ne posait pas de problème car notre directeur aurait eu accès à ce fichier via le réseau d'entreprise : Domaine, système de fichiers, partages, etc…

Imaginons maintenant que le conseil de classe de ces 4 classes se déroulent en même temps. Le directeur fournit le fichier aux 4 titulaires de classe.

Chaque titulaire manipule un fichier car il est possible que l'on donne la moyenne aux élèves proches de la moyenne. C'est tout le débat d'un conseil de classe. Et donc de modifier ce fichier.

Chaque titulaire devra envoyer son fichier au directeur. Et le directeur devra remettre le tout dans un fichier reprenant les modifications de chaque titulaire : aie aie aie sur autant d'élèves les risques d'erreurs commencent à augmenter avec autant de manipulations manuelles.

Idéalement il aurait été très intéressant que chaque titulaire puisse encoder ses modifications et que ces modifications soient prises en compte. Avec un programme, il est plus compliqué de faire la gestion concurrentielle d'un même fichier. Mais ça reste faisable. Mais de nos jours nous avons recours à ce qu'on appelle des bases de données. Le programme se connecte sur une base de données où seraient stockés les résultats de tous les élèves de toutes les classes. L'accès concurrent est généré nativement : chaque titulaire peut modifier en même temps la base de données.

II. Base de données

1. Définition

Une base de données est un outil informatique qui permet d'organiser des informations de façon sécurisée, hiérarchisée et sans doublon. Appelée Database en anglais (on voit souvent l'abréviation db), les bases de données sont des logiciels qui permettent surtout de mieux travailler.

C'est donc une collection structurée de données cohérentes, intègres, protégées et accessibles simultanément aux utilisateurs

Concrètement, les informaticiens se sont rapidement retrouvés face à des problèmes difficiles à résoudre en termes de performance et d'intégrité. Comment s'assurer qu'une information saisie dans un système informatique est unique, toujours bien rangée et correctement protégée contre les mauvaises manipulations ?

Un SGBD peut gérer plusieurs bases. En effet, on pourrait très bien avoir les données des cours d'une école comme base et une autre base pour les données d'un garage de voitures Tesla. C'est tout à fait possible et même utile pour sécuriser et séparer des données différentes.

2. Système de gestion de base de données (SGBD)

Le SGBD n'est que l'application concrète de la base de données. Sans SGBD, la BD reste un outil théorique « sur papier ». Le SGBD permet concrètement de mettre en place le travail de modélisation et de se servir de la base de données imaginée.

Implémenter une base de données dans un SGBD impose d'arrêter son choix sur un outil. Pour le choisir, il faut avoir réfléchi aux contraintes et caractéristiques de la base de données (volume d'information, accès depuis un même lieu ou pas, droits et accès simultanés) … Baser son choix de SGBD uniquement sur les outils disponibles (par exemple Ms Access parce qu'il est installé avec la suite Ms Office) est à coup sûr une mauvaise idée.

Petite parenthèse pour MS Access. Attention Ms Access n'est pas non plus la pire idée du monde. Il reste tout de même fort utilisé dans les asbl, les petites PME car il permet de rapidement faire des formulaires pour ajouter/modifier/supprimer des informations dans une base de données Access. Depuis que MS Access le transactionnel, il n'est plus le choix aussi risible qu'il était par le passé. Tout va dépendre de la charge et du nombre d'utilisateurs.

Un Système de Gestion de Base de Données (SGBD) est un logiciel qui permet de stocker des informations dans une base de données. Un tel système permet de lire, écrire, modifier, trier, transformer ou même imprimer les données qui sont contenus dans la base de données.

De plus, un SGBD est sécurisé via l'utilisation d'utilisateurs, de mots de passe. On peut définir les objets que l'utilisateur pourra utiliser dans notre SGBD : bases, tables, procédures stockées, etc.

Dans le cadre de ce cours, nous utiliserons MySQL comme SGBD. Car elle c'est un logiciel libre. Elle existe en version libre et propriétaire. En version propriétaire et donc payante vous aurez un support de la part de Sun qui a racheté MySQL en 2010.

3. Base de données relationnelles

La révolution apportée par le modèle relationnel réside dans une indépendance totale par rapport au modèle physique. Défini par Codd en 1970 sur des bases purement mathématiques, ce modèle s'affranchit résolument de toute contrainte matérielle. Cela explique qu'il ait démarré assez lentement, parce qu'il exigeait les machines puissantes dont nous disposons seulement aujourd'hui.

4. SQL

Les SGBD offrent un langage d'interrogation des données qui s'appelle le SQL. Le SQL a connu plusieurs normes. On pourrait penser que le SQL est un langage universel et identique à tous SGBD. Et bien non ! En gros oui mais avec parfois des adaptations mineures mais posant un problème majeur : mes commandes SQL ne seront pas exactement les mêmes si je veux changer de base de données. Exemple : passer d'Oracle à MySQL. La volonté de ne pas uniformiser vient du fait qu'un fabricant de SGBD n'a pas envie que vous quittiez son SGBD très facilement. De plus, certains fabricants offrent de belles fonctionnalités par rapport à la concurrence. Donc si vous utilisez des fonctionnalités très spécifiques et propriétaires à un SGBD. Il vous sera difficile de migrer vers un autre SGBD ou au prix de beaucoup d'efforts. Mais pour des requêtes classiques la syntaxe de votre commande SQL sera la même partout.

III. Modèle conceptuel de données (MCD)

Dans la méthodologie Merise destinée à créer des bases de données, il y a des outils dédiés aux traitements et aux données. Le MCD (Modèle Conceptuel des Données) est un des outils majeurs concernant les données. Ce modèle décrit une situation à analyser à l'aide d'entités et de relations. Des entités peuvent être liées à une autre entité par l'intermédiaire de ce qu'on appelle une relation.

Une fois notre modèle fait, on convertira nos entités en tables et nos relations en clés primaires et clés étrangères. Nous verrons plus loin cela.

1. Entité

Une entité a un nom unique afin de la manipuler facilement. Plus tard dans l'analyse, l'entité se transforme en table et devient concrètement une table lors de la réalisation effective de la base de données.

Cet ensemble d'informations, l'entité, partage les mêmes caractéristiques et peut être manipulé au sein du système d'information mais aussi en discutant entre informaticiens et personnes du métier.

Reprenons notre exemple sur les résultats de nos élèves et les notes des élèves pour le cours de python.

Si on devait essayer de rassembler les informations de base d'un élève : de l'entité élève. Quelles sont les propriétés qui le caractérisent et permettent de l'identifier au mieux ?

- Un nom
- Un prénom
- Une date de naissance
- Sexe (F/M/X) : prendre en compte les personnes transgenres. J'ai déjà vu une personne qui avait mis en DB un booléen pour le sexe : EstHomme. On verra plus loin pourquoi ce n'est pas à faire.)
- Un téléphone fixe (ou pas)
- Un gsm (ou pas)
- Email (ou pas)
- Rue

- Numéro d'habitation
- Boîte
- CP
- Ville

Que manque-t-il pour permettre d'identifier de manière unique votre élève ? En effet, certaines données ne sont pas obligatoires : Gsm, téléphone, email.

Pour répondre à cette question. Que demandent les services auxquels vous êtes clients quand vous les appelez ? Ce qu'ils demandent c'est un numéro de client qui vous est propre, unique. Ce numéro vous est attribué. Et ne peut être en double. Dans le cas contraire, ça m'arrangerait qu'une personne ayant le même numéro de client paie mes factures. 😊

Ici dans le cas de notre élève, on pourrait ajouter une propriété pour l'identifier de manière unique : un numéro d'inscription, un matricule, etc. Ce numéro unique peut être un simple numéro incrémenté (incrémenté = augmenté) : 47. Ou bien suivre des règles bien précisent : 1/2021 (Premier étudiant inscrit de l'année 2021).

Chaque propriété de notre élève comporte un type. Reprenons notre entité élève et pour chaque propriété on va typer celle-ci :

Nom du champ	**Type**	**Obligatoire (= PAS NULL)**	**Unique**	**Remarque**
Matricule	Entier	Oui	Oui	C'est le seul champ obligatoire et unique.
Nom	Chaîne de caractères	Oui	Non	
Prénom	Chaîne de caractères	Oui	Non	
Date de naissance	Date	Oui	Non	
Sexe	Caractère	Oui	Non	
Téléphone fixe	Chaîne de caractères	Non	Oui	
Gsm	Chaîne de caractères	Non	Oui	
Email	Chaîne de caractères	Non	Oui	
Rue	Chaîne de caractères	Oui	Non	
Numéro d'habitation	Chaîne de caractères	Oui	Non	
Boîte	Chaîne de caractères	Non	Non	
CP	Entier	Oui	Non	
Ville	Chaîne de caractères	Oui	Non	

La propriété Matricule va s'appeler « clé d'identité ».

2. Clef d'identité

La clef d'identité permet d'identifier de manière sûre et fiable notre élève. Cette clé doit être pensée pour qu'il ne puisse JAMAIS y avoir de doublons. La clef peut être composée d'une ou plusieurs propriétés. Les valeurs de clefs d'identités sont uniques et non nulles.

Dans l'entité, cette clef est soulignée pour marquer justement que c'est une clef.

Il arrive souvent qu'on ajoute le préfixe « Id » (pour identifiant) à une clef d'identité. Exemple : IdClient, IdEtudiant. Cette clef est souvent écrite Id. Rajouter IdEleve dans l'entité Elève est un peu redondant : c'est évident. Mais ça c'est selon l'endroit et les conventions que vous ou votre équipe utiliserez.

Dans beaucoup de cas, c'est souvent un numéro automatique incrémenté de 1. L'intérêt d'avoir un numéro automatique, c'est que la gestion de ce numéro automatique est laissée au SGBD. Il déduira automatique le nouveau numéro à générer pour le nouvel enregistrement. Si le précédent Eleve avait comme identifiant 43, le nouvel étudiant que l'on encodera aura le numéro 44. Et pour le suivant, ça sera le numéro 45, etc.

Souvent on prend un entier comme clef primaire. Car un entier prend 4 octets. Une clef primaire de type entier prend moins de place et la recherche dans les index est plus rapide.

3. Relation/association

La relation ou association relie plusieurs entités.

Notre entité élève fait partie d'une classe. Une classe est une entité.

Ce qui relie l'entité élève et l'entité classe c'est la relation : « fait partie de » dans le sens élève vers classe.

4. Cardinalités/Multiplicité ou combien ?

Dans la théorie des ensembles, la cardinalité est une propriété des ensembles, y compris infinis, qui généralise la notion de nombre d'éléments aux ensembles finis.

Les cardinalités sont des couples de valeurs que l'on retrouve entre chaque entité et ses relations. La première valeur est la valeur minimale et la seconde est la valeur maximale.

Il existe quatre valeurs : (0,1), (0,N), (1,1) ou (1,N) où N >1

Exemple :

- Entité Élève
- Relation : Fait partie de
- Entité Classe

Les cardinalités traduisent des règles de gestion.

Un élève fait partie d'une et une seule classe.

Sens Elève vers Classe : cardinalité 1 (minimum) et 1 (maximum)

Et une classe a 1 ou N (=plusieurs) élèves.

Sens Classe vers Elève : cardinalité 1 (minimum) et N (maximum).

5. Passage du MCD au MLD

Le passage du Modèle Conceptuel de Données (MCD) au Modèle Logique de données (MLD).

Règle n°1

a. Une entité devient une table

Une entité devient une table. Dans un SGBD, une table est une structure composée de colonnes. Ces colonnes sont typées et peuvent avoir des contraintes : unique, non nulle, etc. Ces colonnes correspondent aux propriétés de l'entité. Une colonne porte le nom de champ dans une table. Par exemple le champ « prénom » de la table Elève.

Par convention, on n'utilise pas de caractère accentué pour le nom des champs.

Dans l'affichage de l'ensemble des données de notre table. Par exemple la table Élève qui contient tous les élèves, chaque ligne de cette table correspond à ce qu'on appelle un enregistrement qui correspond à un élève.

La valeur prise par un champ pour un enregistrement donné se situe à l'intersection entre l'enregistrement et le nom du champ.

b. La clef d'identité devient la clef primaire

Dans une table, la clef d'identité devient une clef primaire. La clef primaire (Primary key en anglais) permet d'identifier de manière unique un enregistrement d'une table. Si on liste tous les enregistrements de la table élèves = si on liste tous les élèves de la table élèves, nous n'aurons pas deux élèves avec la même clef primaire. Le SGBD ne le permettrait pas et provoquerait une erreur si on essayait de le faire. La création d'une clef primaire donne lieu dans les SGBD la création d'un index qui permet aux SGBD de traiter plus rapidement les recherches, les tris. C'est très intéressant quand on brasse une très grosse quantité de données.

c. Une propriété devient un attribut

Une propriété d'une entité devient un attribut/champ/une colonne. Ce champ a un type : integer, float, boolean, varchar (chaîne à taille variable), char (chaîne à taille fixe), date (date, datetime, timespamp). Cet attribut peut aussi être qualité de NULL, NOT NULL, UNIQUE par exemple.

Règle n°2

a. Une association de type 1:N

C'est à dire qui a les cardinalités maximales positionnées à « 1 » d'un côté de l'association et à « N » de l'autre côté. Elle se traduit par la création d'une clé étrangère dans la relation correspondante à l'entité côté « 1 ». Cette clé étrangère référence la clé primaire de la relation correspondant à l'autre entité.

<u>Exemple 1</u> :

Elève Fait partie d'une classe : cardinalité 1:1 (Cardinalité maximale = 1)

Et une classe a un ou plusieurs élèves : 1:N (Cardinalité maximale = N)

⇨ Association de type 1:N

On ajoutera dans la table Elève la clef primaire de la table Classe. En effet, cette clé permettra d'identifier la classe dont fait partie un élève. Quand on ajoute comme champ la clef primaire d'une autre table, cette

clef porte le nom de clef étrangère (Foreign Key en anglais). Contrairement à la clef primaire qui doit être unique dans une table, une même valeur de clef primaire peut y figurer plusieurs fois. Ce qui est logique : plusieurs élèves font partie d'une même classe. Elle ne peut être NULL dans ce cas-ci.

Exemple 2 :

On pourrait imaginer que notre système d'inscription autorise l'inscription d'étudiants indécis. Ils ne savent pas ce qu'ils veulent faire dans l'école mais savent qu'ils veulent étudier…

Un élève peut faire partie d'une classe mais peut aussi ne pas en faire partie : cardinalité 0,1 (cardinalité maximale = 1)

Une classe a un ou plusieurs élèves : 1:N (Cardinalité maximale = N)

⇨ Association de type 1 :N

On fera comme précédemment on ajoutera comme clef étrangère, la clef primaire de la table Classe. Mais à la différence qu'ici on acceptera les valeurs NULL pour cette clef étrangère pour nos indécis d'étudiants qui ont par conséquent donnés une cardinalité minimum à 0… 😉

b. Une association de type N :N

C'est-à-dire que les cardinalités maximales des deux entités sont à N.

Dans ce cas précis on doit créer une entité intermédiaire reprenant les deux clefs d'entité. Ces deux clefs d'identité forment alors la clef d'identité de cette nouvelle entité. En effet, à la différence de l'association 1 :N, ici on ne peut avoir qu'une seule valeur mais plusieurs.

Exemple : Cours et Formateur.

Un formateur donne 1 ou plusieurs cours : Cardinalité 1 :N (Cardinalité maximale :N).

Un cours est donné par 1 ou plusieurs cours : Cardinalité 1 : N (Cardinalité maximale :N)

⇨ Association de type N :N

Par exemple le cours de Python est donné par Philip & Johnny.

Philip donne les cours de Python, PHP, Django, Rattrapage, etc.

Johnny donne les cours de Python, Git, SGBD, etc.

Le cours de Python est donné par Philip et Johnny.

Ici on voit bien qu'on ne sait pas mettre qu'une seule clef étrangère Cours dans l'entité Formateur. Il en faudrait plusieurs.

Et on voit bien qu'on ne sait pas mettre qu'une seule clef étrangère Formateur dans l'entité Cours. Il en faudrait plusieurs.

Il faut donc « tout simplement » créer une nouvelle entité qu'on peut appeler FormateurCours. La clef d'identité sera composée de la clef d'identité de l'entité Cours et de la clef d'identité de l'entité Formateur. Cette entité contient donc une clef composite ainsi qu'éventuellement des propriétés propres.

Ma compagne relisant le cours me dit : oui mais pourquoi ne pas créer autant de champs supplémentaires qu'on en a besoin ? Par exemple dans Formateur on mettrait idCours1, idCours2, idCours3, idCours4, etc. Alors j'ai été étonné par sa réflexion mais finalement elle a raison : Pourquoi pas ? Car c'est difficilement maintenable. Imaginons qu'un jour le nombre maximum de cours donné par un formateur passe de 5 à 15 (pauvre formateur !). Ça veut dire qu'on doit modifier l'entité et ajouter 10 propriétés qui sont des clefs étrangères. C'est faisable mais peu évident à maintenir. Par contre notre entité FormateurCours est une solution finale/générique. Cette solution tiendra en compte un nombre infini de cours qu'un formateur pourrait donner (pauvre formateur !).

c. Une association de type 1 :1

Ce type d'association est à proscrire car elle reflète une situation où une entité doit être intégrée dans l'autre entité.

Prenons un exemple : On doit modéliser les entités entrant en jeu dans une course de voiliers en solitaire.

Nous pourrions avoir deux entités : Marin et Voilier avec comme relation Pilote.

Un marin pilote 1 et 1 seul voilier : cardinalité 1:1 (Cardinalité maximale = 1)

Un voilier est piloté par 1 et 1 seul voilier : cardinalité 1:1 (Cardinalité maximale = 1)

⇨ Association de type 1 :1

Si fonctionnellement on considère que Marin est plus important : on ramène toutes les propriétés de Voilier dans Marin.

Si fonctionnellement on considère que Voilier est plus important : on ramène toutes les propriétés de Marin dans Voilier.

Maintenant, si on sait que notre modèle évoluera vers une association de type 1 :N. Ce type d'association pourra gérer par exemple des courses de voiliers. 1 marin pilote 1 et 1 seul voilier. Mais 1 voilier est piloté par 1 ou N marins.

Dans la pratique si les entités ont une distinction fonctionnelle forte. On peut les séparer. En effet, imaginons qu'un voilier ait 100 propriétés qui le caractérisent. Remettre toutes ces propriétés dans l'entité Marin est assez discutable. Personnellement, dans ce cas, je fais deux entités.

IV. Le langage SQL

Nous allons maintenant manipuler les données qui se trouvent dans une base de données. Nous utiliserons un langage qui s'appelle le SQL. Les commandes SQL s'écrivent en MAJUSCULES par convention. Ne pas le faire ne provoquera pas une erreur.

Pour faire simple, voici les commandes de base que l'on utilise en SQL :

1. Chercher des informations avec SELECT

2. Ajouter des enregistrements avec INSERT INTO

3. Modifier des enregistrements avec UPDATE

4. Effacer des enregistrements avec DELETE FROM

Nous allons présenter chaque type de commande SQL au cours de ce chapitre.

1. Chercher des informations : SELECT

L'instruction la plus célèbre du langage SQL est sans conteste l'instruction SELECT. Cette instruction est utilisée pour faire chercher des résultats d'une table ou plusieurs tables.

a. Simple SELECT

Sa forme la plus simple est :

SELECT attr1, attr2, attr3, etc….
FROM NomTable

Ou encore

SELECT *
FROM NomTable

Ici le symbole * prendra tous les attributs de la table en question.

Exemple :

SELECT Nom, Prenom, Sexe, Naissance

FROM Eleve

Cette instruction SQL va nous lister TOUS les élèves de la table Eleve et affichera les attributs Nom, Prenom, Sexe, Naissance. Maintenant, si vous avez 10.000 élèves ça risque d'être le tsunami d'informations 😊 C'est pourquoi on couple avec l'instruction WHERE qui permet de mettre une expression booléenne pour restreindre la quantité d'informations reçues.

b. WHERE : Filtrage simple

WHERE signifie Où en français. Le où indique qu'on attend une condition pour filtrer notre sélection. Seuls les enregistrements répondants à la condition seront affichés.

Exemple :

SELECT Nom, Prenom, Sexe, DateNaissance

FROM Eleve

WHERE Sexe = 'F' ;

Nous afficherons ici tous les élèves qui sont du sexe ‘F’. Le SGBD va parcourir tous les enregistrements et ne garder que les élèves qui rentrent dans la condition Sexe= 'F'.

c. WHERE : Opérateurs Booléens

Les opérateurs booléens pour Mysql dans un WHERE sont

Différent: <> ou !=
Egal: =
Plus grand que : >
Plus grand ou égale : >=
Plus petit que : <
Plus petit ou égale : <=
ET : AND
Ou : OR
Est Null : IS NULL
N’est pas NULL : IS NOT NULL
Par/Comme : LIKE par exemple Nom LIKE 'Pi%' Cherchera les noms commençait par Pi
Entre : BETWEEN Valeur1 AND Valeur2
Dans : IN par exemple : CP IN (6980 , 4000)

d. WHERE : AND et OR

Les opérateurs sont à ajoutés dans la condition WHERE. Ils peuvent être combinés à l’infini pour filtrer les données comme souhaités.

L’opérateur AND permet de s’assurer que la condition1 ET la condition2 sont vrai :

SELECT nom_colonnes

FROM nom_table

WHERE condition1 AND condition2

L’opérateur OR vérifie quant à lui que la condition1 OU la condition2 est vrai :

SELECT nom_colonnes FROM nom_table

WHERE condition1 OR condition2

Ces opérateurs peuvent être combinés à l’infini et mélangés. L’exemple ci-dessous filtre les résultats de la table “nom_table” si condition1 ET condition2 OU condition3 est vrai :

SELECT nom_colonnes FROM nom_table

WHERE condition1 AND (condition2 OR condition3)

Attention : il faut penser à utiliser des parenthèses lorsque c’est nécessaire. Cela permet d’éviter les erreurs car et ça améliore la lecture d’une requête par un humain.

Exemple de données. Ici vous devrez importer la petite base de données : Ventes avec la commande source databaseVentes.sql dans le répertoire exercices du cours.

Pour illustrer les prochaines commandes, nous allons considérer la table “produit” suivante :

IdProduit	Nom	Categorie	Stock	Prix
1	Ordinateur	Informatique	5	950
2	Clavier	Informatique	32	35
3	Souris	Informatique	16	30
4	Crayon	Fourniture	147	2

D1. Opérateur AND

L’opérateur AND permet de joindre plusieurs conditions dans une requête. En gardant la même table que précédemment, pour filtrer uniquement les produits informatiques qui sont presque en rupture de stock (moins de 20 produits disponible) il faut exécuter la requête suivante :

```
SELECT * FROM produit

WHERE categorie = 'informatique' AND stock < 20
```

Cette requête retourne les résultats suivants :

IdProduit	Nom	Categorie	Stock	Prix
1	Ordinateur	Informatique	5	950
3	Souris	Informatique	16	30

D2. Opérateur OR

Pour filtrer les données pour avoir uniquement les données sur les produits “ordinateur” ou “clavier” il faut effectuer la recherche suivante :

```
SELECT * FROM produit

WHERE nom = 'ordinateur' OR nom = 'clavier'
```

Cette simple requête retourne les résultats suivants :

IdProduit	Nom	Categorie	Stock	Prix
1	Ordinateur	Informatique	5	950
2	Clavier	Informatique	32	35

D3. Combiner AND et OR

Il ne faut pas oublier que les opérateurs peuvent être combinés pour effectuer de puissantes recherches. Il est possible de filtrer les produits “informatique” avec un stock inférieur à 20 et les produits “fourniture” avec un stock inférieur à 200 avec la recherche suivante :

```
SELECT * FROM produit

WHERE ( categorie = 'informatique' AND stock < 20 )

OR ( categorie = 'fourniture' AND stock < 200 )
```

Cela permet de retourner les 3 résultats suivants :

IdProduit	Nom	Categorie	Stock	Prix
1	Ordinateur	Informatique	5	950
2	Clavier	Informatique	32	35
4	Crayon	Fourniture	147	2

e. DISTINCT

Si l'on veut connaître par exemple toutes les nationalités de la table élève :

SELECT Nationalite

FROM Eleve

Le problème ici, c'est que l'on va avoir autant de fois Belge que l'on a des étudiants belges. Donc si on a 12 belges et 1 camerounais, on aura comme résultats : 12 fois 'belges' et 1 fois 'camerounais'. Ce qui n'est pas exactement ce que l'on veut.

Pour y arriver, on va utiliser après notre **SELECT** le mot clef **DISTINCT**

SELECT DISTINCT Nationalite

FROM Eleve

Et ici, nous n'aurons plus que 2 résultats : 'Belge' et 'Camerounais'. Dans ce cas, DISTINCT va supprimer les doublons.

f. COUNT(champ1, champ2) / COUNT(*)

Si nous voulons connaître le nombre d'élèves faisant partie de la table Eleve : on utilise COUNT

SELECT COUNT(*)

FROM Eleve ;

On va voir que le résultat sera une colonne avec comme nom : count(*) avec une cellule ayant 13 comme valeur. Il y a donc 13 élèves.

Cependant le nom de la colonne n'est pas très agréable à lire, on peut lui donner un autre nom simple :

SELECT COUNT(*) AS NB_Eleves

FROM Eleve ;

On peut utiliser le mot clef AS qui signifie Comme. On pourrait aussi l'omettre et mettre un espace :

SELECT COUNT(*) NB_Eleves

FROM Eleve ;

Maintenant si nous voulions connaître le nombre de garçons parmi nos élèves nous devons coupler la commande COUNT avec la DISTINCT :

SELECT COUNT (Sexe) AS NB_Garcons

FROM Eleve

WHERE Sexe = 'M ' ;

Nous avons comme résultats 11 garçons. Faisons la même chose avec les filles, nous devrions en avoir : 13-11 = 2 filles

SELECT COUNT (Sexe) AS NB_Filles

FROM Eleve

WHERE Sexe = 'F' ;

Dernier exemple, nous voulons connaître le nombre d'élèves nés de 1990 à aujourd'hui :

SELECT COUNT(*)

FROM ELEVE

WHERE Naissance >= '1995/01/01' ;

g. Tris (clause ORDER BY)

Avoir des données de notre base de données, c'est déjà bien. Mais si en plus le SGBD peut nous les trier selon l'ordre que nous voulons, c'est encore mieux ! C'est là qu'entre en scène ORDER BY.

Soit classer nos élèves par Nom de famille :

SELECT Nom, Prenom, Naissance, Sexe

FROM Eleve

ORDER BY Nom

Classer nos élèves par Nom et puis par prénom : Ce cas est intéressant si nous avons plusieurs mêmes noms de famille, le SGBD classera alors ensuite sur le prénom.

SELECT Nom, Prenom, Naissance, Sexe

FROM Eleve

ORDER BY Nom, Prenom