

BASES DE DONNÉES RELATIONNELLES

Introduction

Les bases de données sont apparues à la fin des années 60, à une époque où la nécessité d'un système de gestion de l'information souple se faisait ressentir.

Dans ce cours, nous allons plus particulièrement aborder la notion de bases de données relationnelles afin de comprendre leur structure et découvrir les moyens de “lire” une telle base de données. Ceci passera par quelques notions de langage SQL, langage de “communication” avec la base de données.

La première partie sera consacrée à la présentation des bases de données relationnelles et notamment des objets fondamentaux que nous allons manipuler : les relations.

Nous étudierons ensuite en deuxième partie les opérations réalisables sur ces relations.

Nous terminerons par quelques compléments afin d'affiner et de mettre en perspective l'utilisation de bases de données.

Première partie

Mise en place du modèle relationnel

1 Concepts généraux

1.1 Base de données

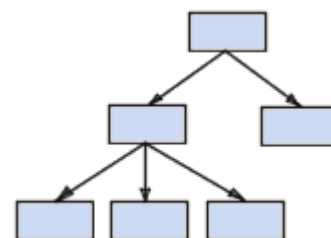
Une base de données informatique est un **ensemble de données** qui ont été stockées sur un support informatique, **organisées et structurées** de manière à pouvoir facilement consulter et modifier leur contenu.

1.2 Différents modèles

Modèle hiérarchique. Dans ce modèle, les données sont classées hiérarchiquement selon une arborescence descendante.

Ce type de modèle de données a été créé dans les années 1960 ; c'est le plus ancien modèle de données.

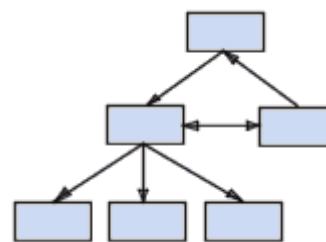
Il est notamment utilisé dans la gestion des fichiers sur un ordinateur.



MODÈLE HIÉRARCHIQUE

Modèle réseau : Ce type de modèle de données est assez semblable au modèle hiérarchique.

La différence porte sur la possibilité de liens “transversaux” ce qui rend ces modèles un peu plus polyvalents.



MODÈLE RÉSEAU

Ces deux modèles présentent tout de même des limites assez importantes quant aux manipulations possibles (recherche ou modification). Une amélioration actuellement très utilisée est le modèle relationnel qui va être l'objet de notre cours.

Modèle relationnel. C'est le type de modèle de données le plus couramment utilisé pour la réalisation d'une base de données.

Selon ce type de modèle, la base de données est composée d'un ensemble de tables (les relations) dans lesquelles sont placées les données ainsi que les liens.

Ces modèles sont simples à mettre en œuvre, ils sont très populaires et fortement normalisés.

Numéro	Nom	Prénom	Email
1	Jean	Dupont	jdupont@email.com
2	Marie	Malherbe	mama@email.com
3	Nicolas	Jacques	Jacques.nicolas@email.com
4	Hadrien	Piroux	happi@email.com

Numéro	Cliant	Produit	Quantité
1	3	Tube de colle	3
2	2	Rame de papier A4	6
3	2	Ciseaux	2

MODÈLE RELATIONNEL

Remarque. Il existe d'autres modèles comme par exemple le modèle *orienté objet* lié à la programmation orienté objet (thème d'étude possible en deuxième année).

1.3 Système de gestion de base de données

Une base de données seule ne suffit pas. Il faut aussi pouvoir la gérer, interagir avec cette base.

Un **Système de Gestion de Base de Données (SGBD)** est un logiciel (ou un ensemble de logiciels) permettant de manipuler les données d'une base de données. Manipuler, c'est-à-dire sélectionner et afficher des informations tirées de cette base, modifier des données, en ajouter ou en supprimer.

On parle de **système de gestion de base de données relationnel (SGBDR)** quand le logiciel implémente le modèle relationnel. C'est le cas de la majorité des SGBD actuels.

Un SGBDR peut fournir une interface graphique ou non. Nous verrons que ce n'est pas toujours nécessaire (et qu'il vaut parfois mieux s'en passer).

Exemples. Il existe de nombreux SGBDR dont IBM DB2, Oracle Database et Microsoft SQL Server (logiciels propriétaires) ou encore PostgreSQL, MySQL et SQLite (logiciels libres), chacun offrant des fonctionnalités différentes.

1.4 Langage SQL

Le SQL (Structured Query Language) est un langage informatique qui permet d'interagir avec des bases de données relationnelles. C'est le langage pour base de données le plus répandu.

Il a été créé dans les années 1970 et est devenu standard en 1986 - 1987. Il est encore régulièrement amélioré.

Chaque instruction transmise, appelée **requête** (query en anglais), se construit à partir de mot-clés que nous découvrirons au fur et à mesure de l'avancée du cours.

2 Vers l'algèbre relationnelle

2.1 Représentation dans le modèle relationnel

Définition (Relation).

Dans un SGBDR, les données sont contenues dans ce qu'on appelle des relations, qui sont représentées sous forme de **tables**.

Une table est composée de deux parties, l'en-tête et le corps.

- L'en-tête est lui-même composé de plusieurs **attributs**. La liste de ces attributs constituent le **schéma relationnel** de la table.
- Quant au corps, il s'agit d'un ensemble de n -uplets, appelés **enregistrements ou valeurs**, composées d'autant d'éléments qu'il y a d'attributs dans l'en-tête.

L'ensemble de ces n -uplets constitue **la relation** proprement dite.

A chaque attribut est associé un **domaine** *i.e.* un ensemble de valeurs possibles.

Exemple. La papeterie "Au stylo rouge" (spécialiste en stylo rouge) utilise une relation/-table "Clients" de schéma relationnel

Numéro	Prénom	Nom	Email
--------	--------	-----	-------

On a donc 4 attributs.

Le domaine de l'attribut **Numéro** est

Celui des autres attributs serait

Voici ci-dessous une table possible de clients (fictifs bien entendu).

Numéro	Prénom	Nom	Email
1	Jean	Dupont	jdupont@email.com
2	Marie	Malherbe	mama@email.com
3	Nicolas	Jacques	Jacques.nicolas@email.com
4	Hadrien	Piroux	happi@email.com
5	Mathurin	Ematic	mathematic@email.com

La table contient 5 lignes (les enregistrements ou valeurs).

Notation. Soit R une relation et A un attribut de cette relation.

Si e est un enregistrement de cette relation, on note $e.A$ la valeur de l'attribut A dans e .

Exemple. On reprend la table **Clients** précédente pour laquelle on note $e1, e2, e3, e4, e5$ les 5 enregistrements.

La notation $e2.Prénom$ désigne

La notation $e4.Email$ désigne

2.2 Description de l'algèbre relationnelle

Nous venons de présenter les objets que nous allons manipuler : les relations.

Il nous reste maintenant (dans la partie suivante) à décrire les actions que nous allons pouvoir effectuer : les opérateurs.

Ces opérateurs se classent en deux grandes familles :

Opérateurs ensemblistes. Ce sont en fait des opérateurs usuels (à une exception près) sur les ensembles vus dans un contexte de bases de données.

Opérateurs dits spécifiques. Ces opérateurs exploitent la structure sous-jacente des relations (le schéma relationnel) et notamment les attributs des relations.

Nous allons les étudier en deux temps : sélections et projections tout d'abord ; jointures et renommages ensuite.

L'**algèbre relationnelle** se constitue de l'ensemble $\mathcal{R}$ des relations munis des opérateurs énumérés ci-dessus.

Deuxième partie

Opérateurs relationnels

3 Opérateurs spécifiques (première série)

3.1 Projection

Le but de la projection est de choisir certaines colonnes de la table.

Définition (Projection d'une relation). Soit R une relation de schéma S (la liste de ses attributs).

On considère une sous-famille $(A_1, \dots, A_p)$ d'attributs de S .

On définit alors $\pi_{(A_1, \dots, A_p)}(R)$ la **projection de R selon $(A_1, \dots, A_p)$** comme l'ensemble des relations de la forme $(e.A_1, \dots, e.A_p)$.

$$\text{Symboliquement} \quad \pi_{(A_1, \dots, A_p)}(R) = \{(e.A_1, \dots, e.A_p) \mid e \in R\}$$

Le schéma de cette nouvelle relation est alors $(A_1, \dots, A_p)$.

Exemple. Reprenons la table Clients précédente

La projection $\pi_{(\text{Prénom}, \text{Nom})}(\text{Clients})$ est la nouvelle table ci-contre.

Son schéma est $(\text{Prénom}, \text{Nom})$ uniquement.

Prénom	Nom
Jean	Dupont
Marie	Malherbe
Nicolas	Jacques
Hadrien	Piroux
Mathurin	Ematic

3.2 Sélection

Le but de la sélection est de choisir certaines lignes de la table.

Définition (Sélection d'une relation selon une égalité). Soit R une relation, A un attribut donné de R et a une valeur possible dans le domaine de A .

On définit alors $\sigma_{A=a}(R)$ la **sélection de R suivant la condition $A = a$** comme l'ensemble des enregistrements e de R vérifiant la condition $e.A = a$.

Symboliquement $\sigma_{A=a}(R) = \{e \in R \mid e.A = a\}$

Remarque. Lorsque cela a un sens, on peut étendre cette définition à des conditions plus complexes : inégalités, conditions faisant intervenir plusieurs attributs, utilisation d'opérateurs logiques (ET, OU,...).

3.3 Exemples

Notre papeterie utilise la table **Achats** ci-dessous.

Achats			
Numéro_achat	Numéro_Client	Produit	Quantité
1	3	Tube de colle	3
2	2	Ramette papier A4	6
3	2	Ciseaux	2

- a. Pour obtenir les achats effectués par le client numéro 2, on effectue la sélection

$\sigma_{\text{Numéro_Client}=2}(\text{Achats})$

Numéro_achat	Numéro_Client	Produit	Quantité
2	2	Ramette papier A4	6
3	2	Ciseaux	2

- b. Pour obtenir les achats d'au moins 3 produits identiques, on effectue la sélection

$\sigma_{\text{Quantité} \geq 3}(\text{Achats})$

Numéro_achat	Numéro_Client	Produit	Quantité
1	3	Tube de colle	3
2	2	Ramette papier A4	6

- c. Parmi cette sélection, on ne veut que les produits concernés et leur quantité.
Dans ce cas, on peut composer projection et sélection comme suit.

$\pi_{(\text{Produit}, \text{Quantité})}(\sigma_{\text{Quantité} \geq 3}(\text{Achats})) :$	Produit	Quantité
	Tube de colle	3
	Ramette papier A4	6

3.4 Traduction en langage SQL

3.4.1 Écriture générique

Par souci de concision et de clarté, les deux opérations de projection et de sélection se traduisent en langage SQL au moyen du même **mot-clé SELECT** sous la forme générique suivante :

```
SELECT Att1 , ... , Attp FROM table WHERE condition ;
```

3.4.2 Écritures particulières

Pas de condition. La clause WHERE est bien sûr à enlever s'il n'y a pas de condition...

Joker *. Si l'on veut tous les attributs, il n'est pas obligatoire de les lister. On peut les remplacer par le "joker" *

Condition complexe. On peut utiliser les mots-clés AND et OR dans la condition

3.4.3 Exemples

Reprenons les exemples de la section 3.3

a. La sélection $\sigma_{\text{Numéro_Client}=2}(\text{Achats})$ se traduit par la requête

b. La sélection $\sigma_{\text{Quantité} \geq 3}(\text{Achats})$ se traduit par la requête

c. La composition projection et sélection $\pi_{(\text{Produit}, \text{Quantité})}(\sigma_{\text{Quantité} \geq 3}(\text{Achats}))$ se traduit par la requête

Remarque (SELECT, le faux-ami). Ainsi le mot-clé SELECT effectue en fait une projection et c'est la clause WHERE qui effectue la sélection proprement dite...

UN PEU DE SYNTAXE SQL

- **Le langage SQL n'est pas sensible à la casse.** Autrement dit utiliser des lettres capitales ou non lui est indifférent pour les mot-clés et les noms de tables ou d'attributs.
Cependant, l'usage est d'écrire les mots-clés entièrement en capitales afin de mieux lire la structure de la requête.
- Attention toutefois au fait que ceci n'est pas valable pour les éventuelles chaînes de caractères utilisées (classiquement délimités par des apostrophes).
Ainsi pour notre exemple, écrire comme condition `Produit='Tube de colle'` ou bien `Produit ='tube de colle'` ne donnera pas le même résultat (en fait aucun résultat dans le deuxième cas...).
- *Les noms de table ou d'attributs ne nécessitent pas de délimiteurs tant qu'ils ne comportent pas d'espaces.*
Autant respecter cette habitude (utilisez des tirets bas pour remplacer les espaces).
- **Chaque requête SQL se termine par un point-virgule.**

4 Opérateurs ensemblistes

On a vu qu'une relation était un *ensemble* de n -uplets où n est le nombre d'attributs.

Les opérateurs ensemblistes vont agir sur des relations en les voyant uniquement comme ensembles et sans se préoccuper des noms des attributs ou de leur signification.

4.1 Opérateurs ensemblistes élémentaires

Étant données deux relations ayant même nombre d'attributs, on peut leur appliquer les opérateurs ensemblistes usuels que sont la réunion, l'intersection ou la différence.

4.1.1 Rappel des définitions

Union. L'union de deux relations $R_1 \cup R_2$ est l'ensemble de tous les enregistrements présents dans R_1 **ou** R_2 .

Intersection. L'intersection de deux relations $R_1 \cap R_2$ est l'ensemble de tous les enregistrements présents dans R_1 **et** R_2 .

Différence. La différence de deux relations $R_1 - R_2$ est l'ensemble de tous les enregistrements présents dans R_1 **mais pas** dans R_2 .

4.1.2 Traduction en langage SQL

Union. On utilise le mot-clé UNION.

Intersection. On utilise le mot-clé INTERSECT.

Différence. On utilise les mots-clés EXCEPT ou MINUS (suivant le SGBD ; à vous de tester...).

4.1.3 Exemples

Reprenons la table **Clients** de notre papeterie.

1	Jean	Dupont	jdupont@email.com
2	Marie	Malherbe	mama@email.com
3	Nicolas	Jacques	Jacques.nicolas@email.com
4	Hadrien	Piroux	happi@email.com
5	Mathurin	Ematic	mathematic@email.com

Cette papeterie décide de fusionner avec la papeterie voisine qui a sa propre table de clients appelée `Clients_bis`.

1	Jean	Dupont	jdupont@email.com
2	Julien	Grand	jg@email.com
3	Mathurin	Ematic	mathematic@email.com

- a. Pour connaître l'ensemble des clients des deux papeteries on tente d'effectuer la requête

```
SELECT * FROM Clients
UNION
SELECT * FROM Clients_bis ;
```

de résultat

1	Jean	Dupont	jdupont@email.com
2	Marie	Malherbe	mama@email.com
3	Nicolas	Jacques	Jacques.nicolas@email.com
4	Hadrien	Piroux	happi@email.com
5	Mathurin	Ematic	mathematic@email.com
2	Julien	Grand	jg@email.com
3	Mathurin	Ematic	mathematic@email.com

Remarques.

- L'union élimine automatiquement les doublons (voir par exemple Jean Dupont).
- Cependant les lignes entières doivent coïncider. Ainsi Mathurin ayant deux numéros de client différents apparaît deux fois.

Quelle requête permettrait d'éviter ce désagrément ?

...

...

...

- b. On souhaite déterminer les clients communs aux deux papeteries (attention à Mathurin).

Donner une requête possible :

...

...
...

- c. Donner maintenant une requête permettant d'obtenir les clients de la papeterie voisine qui ne sont pas clients de la papeterie "Au stylo rouge"

...
...
...

4.1.4 Une mise en garde

Les opérations élémentaires ensemblistes peuvent s'effectuer dès que les nombres d'attribut sont les mêmes et ce indépendamment de toute signification et même indépendamment du domaine des attributs.

Ainsi SQL peut tout à fait effectuer l'union des tables **Clients** et **Achats** mais cela n'aurait pas beaucoup de sens pour l'utilisateur... En particulier la dernière colonne mélangerait entiers (les quantités) et chaînes de caractères (les adresses mail).

4.2 Opérateurs ensemblistes complexes

Pour deux relations quelconques, les opérateurs élémentaires ne pourront pas toujours s'appliquer. En revanche, on pourra toujours considérer leur produit cartésien et leur division cartésienne.

Ces deux notions sont assez abstraites et sont rarement utilisées telles quelles.

4.2.1 Produit cartésien

Définition. Soient R_1 et R_2 deux relations.

Les éléments du produit cartésien $R_1 \times R_2$ sont les concaténations d'un élément quelconque de R_1 et d'un élément quelconque de R_2 .

Symboliquement

$$R_1 \times R_2 = \left\{ (v_1, \dots, v_n, w_1, \dots, w_p) \mid (v_1, \dots, v_n) \in R_1; (w_1, \dots, w_p) \in R_2 \right\}$$

Traduction SQL. Le produit cartésien de deux tables s'obtient par la requête

```
SELECT * FROM table1, table2 ;
```

Exemple. Voici le produit cartésien des tables **Clients** et **Achats**

Numéro	Prénom	Nom	Email	Numéro_achat	Numéro_Client	Produit	Quantité
1	Jean	Dupont	jdupont@email.com	1	3	Tube de colle	3
1	Jean	Dupont	jdupont@email.com	2	2	Ramette papier A4	6
1	Jean	Dupont	jdupont@email.com	3	2	Ciseaux	2
2	Marie	Malherbe	mama@email.com	1	3	Tube de colle	3
2	Marie	Malherbe	mama@email.com	2	2	Ramette papier A4	6
2	Marie	Malherbe	mama@email.com	3	2	Ciseaux	2
3	Nicolas	Jacques	Jacques.nicolas@em...	1	3	Tube de colle	3
3	Nicolas	Jacques	Jacques.nicolas@em...	2	2	Ramette papier A4	6
3	Nicolas	Jacques	Jacques.nicolas@em...	3	2	Ciseaux	2
4	Hadrien	Piroux	happi@email.com	1	3	Tube de colle	3
4	Hadrien	Piroux	happi@email.com	2	2	Ramette papier A4	6
4	Hadrien	Piroux	happi@email.com	3	2	Ciseaux	2
5	Mathurin	Ematic	mathematic@email....	1	3	Tube de colle	3
5	Mathurin	Ematic	mathematic@email....	2	2	Ramette papier A4	6
5	Mathurin	Ematic	mathematic@email....	3	2	Ciseaux	2

UN PRODUIT CARTÉSIEN

Remarque. Les lignes d'un produit cartésien ont peu de sens telles quelles. Nous verrons bientôt comment s'en débarrasser.

4.2.2 Division cartésienne

Un rappel d'arithmétique. Si a et b sont deux naturels non nuls, le quotient q de la division euclidienne de a par b est le plus grand entier respectant la condition $b \times q \leq a$.

On s'inspire de ce fait pour définir la division cartésienne.

Définition. Soient A et B deux relations.

La division cartésienne de A par B est la relation Q la plus grande (pour l'inclusion) vérifiant $B \times Q \subset A$.

Remarque. La division cartésienne n'existe qu'en théorie pour s'assurer qu'il soit possible d'exprimer toutes les requêtes possibles. En pratique elle ne s'exprime pas en langage SQL.

5 Opérateurs spécifiques (deuxième série)

5.1 Jointures symétriques

Outil fondamental des bases de données relationnelles, la jointure (comme son nom l'indique) permet de joindre des tables en respectant une condition donnée.

5.1.1 Le principe

Revenons sur notre exemple de produit cartésien vu précédemment.

Parmi toutes les lignes qu'il comporte celles qui vérifient $\text{Numéro} = \text{Numéro_Client}$ nous informent sur un acheteur et son achat. Les autres n'ont pas de signification particulière.

Il serait donc pertinent d'effectuer une sélection selon cette condition pour obtenir

$$\sigma_{\text{Numéro}=\text{Numéro_Client}}(\text{Clients} \times \text{Achats})$$

Numéro	Prénom	Nom	Email	Numéro_achat	Numéro_Client	Produit	Quantité
2	Marie	Malherbe	mama@...	2	2	Ramette...	6
2	Marie	Malherbe	mama@...	3	2	Ciseaux	2
3	Nicolas	Jacques	Jacques....	1	3	Tube de ...	3

5.1.2 Un peu de théorie

Définition (Jointure symétrique). Soient R_1 et R_2 deux relations, A_1 un attribut de R_1 et A_2 un attribut de R_2 (A_1 et A_2 supposées distincts).

La **jointure symétrique** de R_1 et R_2 selon la clause $A_1 = A_2$ est

$$R_1 \bowtie_{A_1=A_2} R_2 = \sigma_{A_1=A_2}(R_1 \times R_2)$$

Remarque. Les colonnes correspondant aux attributs A_1 et A_2 étant identiques après jointure, on effectuera quasi-systématiquement une projection pour éliminer cette répétition.

5.1.3 Traduction SQL

La syntaxe générique d'une jointure est la suivante

```
SELECT *
FROM table1 JOIN table2
      ON Att1 = Att2 ;
```

Remarques.

- On peut bien sûr préciser des noms explicites de colonnes au lieu de `*`.
- Les indentations et sauts de ligne sont facultatifs en SQL... mais fortement conseillés ! En effet lorsqu'une requête mélange plusieurs noms d'attributs et des conditions complexes, cela améliore grandement la lisibilité.
- Il est théoriquement possible d'utiliser la requête

```
SELECT * FROM table1, table2 WHERE Att1 = Att2
```

Cependant cela n'est pas toujours conseillé, notamment pour des raisons de lisibilité (confusion entre les "vraies" sélections et les jointures).

Noms d'attributs identiques. Il peut arriver que les noms d'attributs sur lesquels la condition doit porter soient identiques. Dans ce cas on peut utiliser la notation complète de l'attribut (cf 2.1) sous la forme `table.Att`.

5.1.4 Exemples

On redonne les schémas relationnels des tables `Clients` et `Achat`.

Clients :

Numéro	Prénom	Nom	Email
--------	--------	-----	-------

Achat :

Numéro_achat	Numéro_Client	Produit	Quantité
--------------	---------------	---------	----------

- a. Écrire une requête SQL donnant pour chaque achat effectué dans la papeterie le prénom et le nom de l'acheteur ainsi que le produit acheté.

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

- b. La papeterie voisine utilise une table `Achats_bis` où le numéro de client est simplement appelé `Numéro`.

Clients_bis :

Numéro	Prénom	Nom	Email
--------	--------	-----	-------

Achat_bis :

Numéro_achat	Numéro	Produit	Quantité
--------------	--------	---------	----------

Écrire une requête SQL donnant pour chaque achat effectué dans la papeterie le prénom et le nom de l'acheteur ainsi que le produit acheté.

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

5.2 Renommage

5.2.1 Définition

Cette opération, qui porte bien son nom, consiste à renommer certains attributs.

Cela peut être fait pour alléger une requête complexe ou pour lever une ambiguïté dans un résultat.

Définition (renommage). Soit R une relation de schéma $S = (A_1, \dots, A_n)$ et A_i un attribut donné.

On se donne un nouvel attribut B_i de même domaine que A_i .

La relation obtenue par **renommage** de A_i en B_i est définie par :

- le nouveau schéma obtenu $S' = (A_1, \dots, A_{i-1}, B_i, A_{i+1}, \dots, A_n)$
- les mêmes enregistrements que R .

Elle est notée $\rho_{A_i \leftarrow B_i}(R)$.

Remarque. Par extension on peut définir $\rho_{A_1 \leftarrow B_1, \dots, A_n \leftarrow B_n}(R)$ la relation obtenue par renommage des attributs $A_1, \dots, A_n$.

5.2.2 Traduction SQL

Le renommage s'effectue au moyen du mot-clé **AS** sous la forme suivante **A AS B**.

Exemple. Pour lever l'ambiguïté concernant l'attribut **Numéro** de la relation **Achats_bis** on peut utiliser la requête **SELECT Numéro AS Numero_Client FROM Achats_bis**

Remarque. Le renommage peut aussi s'effectuer avec la même syntaxe sur un nom de table (dans une clause **FROM** par exemple).

Troisième partie

Compléments

6 Fonctions d'agrégation

6.1 Le principe

Définition. L'agrégation consiste à regrouper les enregistrements d'une table selon certains critères. Ces regroupements sont appelés **agrégats**.

On peut ensuite appliquer à ces agrégats certaines fonction dites **fonction d'agrégation**.

Exemple. Notre papeterie complète sa table **Achats** comme suit

Achats				
Numéro_achat	Numéro_Client	Produit	Quantité	Montant
1	3	Tube de colle	3	7,50
2	2	Ramette papier A4	6	39
3	2	Ciseaux	2	5,20

On peut vouloir regrouper les lignes suivant l'attribut **Numéro_Client** puis déterminer la somme sur l'attribut **Montant** pour obtenir le total des dépenses de chaque client.

Remarque (sélection suivant une fonction d'agrégation). Dans certains cas on veut utiliser une condition écrite au moyen d'une fonction d'agrégation pour sélectionner certains enregistrements.

Par exemple déterminer les clients ayant dépensé au moins 10 Euros.

6.2 Traduction SQL

6.2.1 Application d'une fonction d'agrégation

Un regroupement suivi de l'application d'une fonction d'agrégation s'effectue au moyen de la requête suivante

```
SELECT f(Att1), [...] FROM table GROUP BY Att2 ;
```

où f est une fonction d'agrégation

QUELQUES FONCTIONS D'AGRÉGATION EN SQL

SUM(Att)	Somme des champs Att des lignes du groupe
MAX(Att)	Maximum des champs Att des lignes du groupe
MIN(Att)	Minimum des champs Att des lignes du groupe
AVG(Att)	Moyenne des champs Att des lignes du groupe
COUNT(*)	Nombre de lignes du groupe

Remarques.

- Il est tout à fait possible de lister dans le même **SELECT** d'autres attributs (affectés ou non d'une fonction d'agrégation).
- La clause **GROUP BY** peut être omise si on veut appliquer une fonction à toute la table.
- La fonction **COUNT** comptant simplement les lignes s'utilise généralement avec le joker *****.

Exemple. Obtenir le total des dépenses de chaque client s'obtient par

```
SELECT SUM(Montant) AS Total, Numéro_client
FROM Achats
GROUP BY Numéro_Client ;
```

Total	Numéro_Client
44,2	2
7,5	3

Remarques. • Le renommage est souvent utilisé pour obtenir des noms de colonnes plus explicites.

- Ne pas oublier de prendre **Numéro_client** aussi. Sinon on n'obtiendrait que la première colonne ce qui ne serait pas très parlant...
Au passage, quelle opération serait nécessaire pour obtenir le Nom du client au lieu du Numéro ? ...

6.2.2 Sélection suivant une fonction d'agrégation

Une telle sélection s'effectue au moyen du mot-clé **HAVING**.

```
SELECT A1, [...] FROM table GROUP BY Att1 HAVING f(Att2)op_comp Valeur ;
```

où **op_comp** est à remplacer par un opérateur de comparaison : **=**, **<**, **>**, **<=**, **>=**

Remarque.

Exemple. Pour obtenir les numéros des clients ayant dépensé au moins 10 Euros on écrirait

```
SELECT Numéro_Client, SUM(Montant) AS Somme
FROM Achats
GROUP BY Numéro_Client
HAVING SUM(Montant) >= 10 ;
```

6.3 Exercices

- a. Écrire une requête SQL donnant le numéro de chaque client et la moyenne des montants de ses achats.

...
...
...

- b. Écrire une requête SQL donnant pour chaque client *son nom* et le nombre d'achats effectués.

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

- c. Écrire une requête SQL donnant pour chaque client *son nom* et le montant maximal dépensé lors d'un achat à condition qu'il ait effectué au moins 2 achats.

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

7 Clés d'une relation

Exemple (introductif). Vous avez peut-être constaté dans nos exemples que les attributs **Numéro_...** jouaient un rôle particulier lors de l'identification d'une ligne ou lors de jointures.

En effet ils permettaient d'identifier de manière unique :

- une ligne de la table en question (comme **Numéro** de la table **Clients**)
- ou une ligne d'une autre table lors d'une jointure (comme **Numéro_Client** de la table **Achats**)

Ces considérations nous mènent aux notions de clés primaires et étrangères.

7.1 Clé primaire

Définition. Une clé primaire est un ensemble d'attributs (souvent un seul) qui permet d'identifier de manière unique chaque ligne de la table. Les attributs correspondant ne peuvent pas être laissés vides.

Exemple. Pour la table **Clients**, l'attribut **Numéro** peut jouer le rôle de clé primaire.

En revanche l'ensemble (**Nom**, **Prénom**) ne le peut pas car deux personnes peuvent avoir les mêmes nom et prénom.

Et l'email ? Il est impossible que deux personnes aient la même adresse email, donc la contrainte d'unicité semble respectée. Mais que fait-on si une personne n'a pas d'email ?

Remarques.

- On peut préciser dans la base de données que tel ensemble d'attributs est à considérer comme clé primaire. Cela permet au SGBD de vérifier la contrainte d'unicité et la contrainte "non vide" lorsqu'on rentre un enregistrement.
- Souvent une clé primaire est en fait un nombre entier introduit pour la circonstance. Dans ce cas on peut demander au SGBD de le remplir lui-même par "auto-incrémentation".

7.2 Clé étrangère

Définition (simplifiée). Pour nous une clé étrangère sera un attribut (ou ensemble d'attributs) d'une table faisant référence à une clé primaire d'une autre table.

Exemple. Dans la table **Achats**, l'attribut **Numéro_Client** fait référence à l'attribut **Numéro** de la table **Clients**. Il peut être défini comme clé étrangère.

Remarques. Lorsqu'un attribut *A* est défini comme clé étrangère en référence à un attribut *B*, on ne peut rentrer comme valeur de *A* qu'une valeur déjà existante de *B*.

On parle de *contrainte d'intégrité*.

8 Architecture trois-tiers

Vous ne le savez peut-être pas mais vous utilisez des bases de données très fréquemment (recherche internet, accès à un forum, récupération d'un programme de colle, ...).

Pour autant vous n'avez probablement jamais eu besoin d'écrire une requête SQL.

En fait c'est l'application utilisée qui fait l'intermédiaire entre l'utilisateur et la base de données. On parle d'**architecture trois-tiers** :

- le tiers utilisateur (vous, ou plus précisément votre ordinateur)
- le tiers applicatif (l'interface que vous utilisez, un logiciel ou une page internet par exemple)
- le tiers base de données (que vous ne voyez jamais).

La communication entre les tiers passe le plus souvent par un réseau.

Exemple. Lorsque vous accédez à la page mathématiques du site `pcsi.lamartin.fr`, le site internet questionne une base de données pour obtenir la liste des programmes de colle avec un requête du type

```
SELECT id, date_debut, [...] AS date
FROM programmes WHERE matiere= maths
```

que vous êtes maintenant en mesure de comprendre parfaitement.