Chapitre 1

Bases de données : Théorie

« Les données sont partout et il faut les stocker »

À l'ère du *big data* (**données massives** en français), il est plus que jamais nécessaire de stocker correctement des informations.

Stocker *correctement* ne signifie pas choisir un bon modèle de disque dur ou dupliquer les données afin d'en éviter les pertes, mais plutôt stocker en pensant au traitement qui sera appliqué à ces informations : sélection, filtrage, tri...

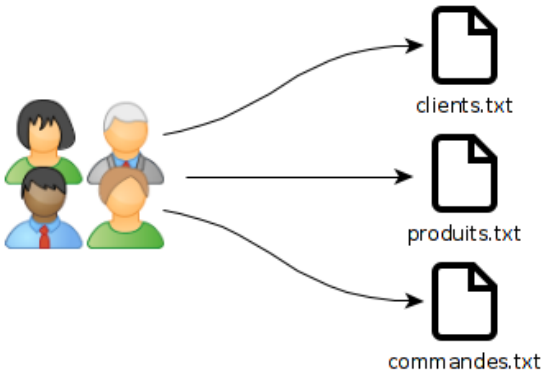
Il faut également penser au partage de ces données, qui se fait aujourd'hui à travers Internet et ses millions d'utilisateurs.

Nous avons déjà appris à utiliser des fichiers pour lire et enregistrer des informations, mais nous allons voir que pour des données conséquentes, le stockage « brut » dans un fichier est rapidement dépassé.

Problème. *Comment **stocker**, **manipuler** et **partager** un volume d'informations toujours plus important ?*

1.1 Les limites du stockage dans un fichier

Imaginons la situation suivante : vous développez un logiciel de gestion clients pour une entreprise, et vous enregistrez les informations sur ces clients dans un fichier *clients.txt*. La liste des produits disponibles à la vente est disponible dans le fichier *produits.txt*, et les commandes passées par ces clients sont enregistrées dans un fichier *commandes.txt*.
Ces fichiers sont accessibles sur le réseau local de l'entreprise, afin que chaque employé y ait accès.



Problème. *Un employé de l'entreprise décide d'ajouter un client. Comme vérifier que ce client n'existe pas déjà ?*

C'est facile, il vous suffit de programmer une fonction dont le rôle est de vérifier l'existence d'un client. Typiquement, vous allez parcourir entièrement le fichier *clients.txt* à la recherche du nouveau client, pour voir si ce dernier n'existe pas déjà...

Problème. *Un employé décide d'ajouter un nouveau produit disponible à la vente. Comment vérifier que ce produit n'est pas déjà enregistré ?*

Encore plus facile : il vous suffit de programmer une fonction sensiblement identique à celle qui vérifie l'existence d'un client.

▷ ***Deux fonctions pour un rôle quasi-identique $\Longrightarrow$ redondance du code $\Longrightarrow$ risque d'erreurs***

Problème. *Un employé ajoute une nouvelle commande passée par un client, en utilisant une fonction spécialement développée à cet effet. À quelles vérifications cette fonction doit-elle procéder ?*

En premier lieu, il faut vérifier l'existence du client, puis l'existence de tous les produits de la commande. Ensuite, il faut vérifier que les produits sont disponibles en quantité suffisante.

▷ ***Nécessité de gérer les relations qui existent entre les 3 fichiers $\Longrightarrow$ risque d'erreurs***

Problème. *Un employé passe une commande dans laquelle un produit n'est plus disponible qu'en un seul exemplaire. Au même moment, un second employé passe une commande différente mais contenant le même produit. Quelle commande est enregistrée ?*

Le problème est difficile, car il fait intervenir la notion de *simultanéité*. Vous réfléchissez un peu... Il faudrait simplement recalculer les quantités disponibles lors de la validation de la commande... Mais empêcher la prise de toute autre commande lors de cette phase de vérification... Bref, c'est compliqué !

▷ ***Problème de gestion des accès concurrents***

Problème. *Un employé est en train de modifier une fiche client lorsqu'une panne de courant survient. Quelles données sont enregistrées ?*

Il faut gérer les pannes de courant en plus ? Et puis quoi encore ?

▷ ***Problème de reprises sur pannes***

Stocker des données dans des fichiers n'apparaît donc pas comme une bonne solution.
En effet, il incombe alors au programmeur la gestion :

- De la **structure** des données
- De la **cohérence** des données
- Des **algorithmes** de recherche des données
- Des problèmes de **concurrence** (accès multiples)
- Des problèmes de **pannes**

Pour toutes ces raisons, il est ainsi préférable d'utiliser des systèmes disposant de fonctions déjà codées, s'adaptant à tous types de problèmes, que le programmeur utilisera sans avoir besoin de tout re-coder.

☞ *Pour de petits projets, le stockage du données dans des fichiers est préférable. C'est le cas par exemple si l'on souhaite enregistrer la liste des 10 meilleurs score dans un jeu. Inutile d'utiliser une base de données !*

1.2 Système de Gestion de Bases de Données

1.2.1 Le principe

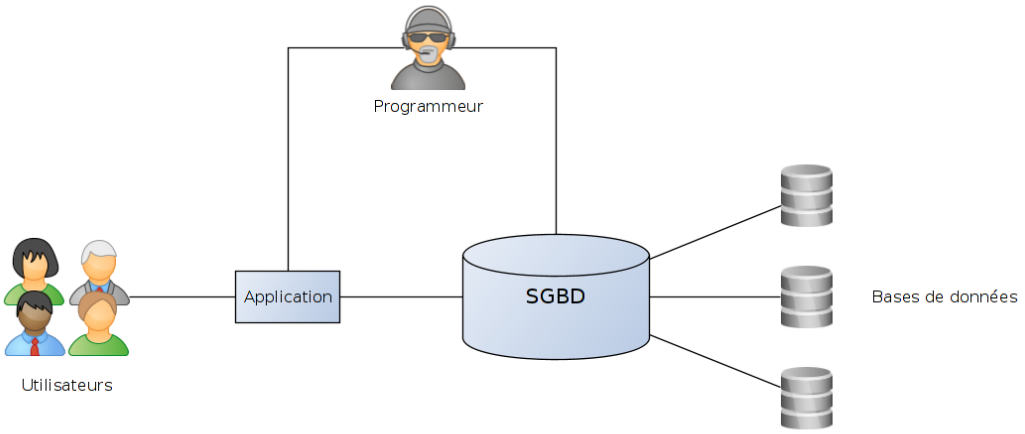
Définition. Un **Système de Gestion de Bases de Données** (SGBD) est un outil informatique permettant la sauvegarde, l'interrogation, la recherche, la mise à jour et le contrôle des données.

On peut utiliser un SGBD depuis une interface graphique ou bien grâce à un langage textuel spécifique.
Comme son nom l'indique, un SGBD permet la manipulation de **bases de données** qui sont des « conteneurs » de données.

En plus des fonctions principales citées dans la définition, un SGBD :

- assure le partage des données
- vérifie qu'une opération s'effectue entièrement ou pas du tout : on appelle cela l'*atomicité*
- optimise les performances (temps de recherche minimisé par exemple)

En général, un utilisateur (comme un employé de notre entreprise) n'utilise pas directement le SGBD mais passe plutôt par une application (comme le logiciel de gestion de clients) qui dialogue directement avec le SGBD.



Le programmeur intervient au niveau de l'application et du SGBD : c'est lui qui programme les interactions entre l'application et le SGBD, et c'est lui qui est amené à définir la structure des données via le SGBD.

Mais personne n'intervient directement sur les bases de données : on passe toujours à minima par le SGBD pour y modifier les informations.

1.2.2 Modèles de données

Il est nécessaire de préciser le **modèle de données** déterminant la structure de nos bases de données.
Parmi les modèles existants, on recense :

- les modèles **hiérarchiques** et les modèles **réseau** (les « vieux » modèles)
- les modèles **relationnels** (les plus utilisés)
- les modèles NoSQL

Chaque modèle dispose de ses avantages et de ses inconvénients. Voici une comparaison de la popularité des SGBD :

Rank			DBMS	Database Model	Score		
Sep 2020	Aug 2020	Sep 2019			Sep 2020	Aug 2020	Sep 2019
1.	1.	1.	Oracle	Relational, Multi-model	1369.36	+14.21	+22.71
2.	2.	2.	MySQL	Relational, Multi-model	1264.25	+2.67	-14.83
3.	3.	3.	Microsoft SQL Server	Relational, Multi-model	1062.76	-13.12	-22.30
4.	4.	4.	PostgreSQL	Relational, Multi-model	542.29	+5.52	+60.04
5.	5.	5.	MongoDB	Document, Multi-model	446.48	+2.92	+36.42
6.	6.	6.	IBM Db2	Relational, Multi-model	161.24	-1.21	-10.32
7.	7.	↑ 8.	Redis	Key-value, Multi-model	151.86	-1.02	+9.95
8.	8.	↓ 7.	Elasticsearch	Search engine, Multi-model	150.50	-1.82	+1.23
9.	9.	↑ 11.	SQLite	Relational	126.68	-0.14	+3.31
10.	↑ 11.	10.	Cassandra	Wide column	119.18	-0.66	-4.22

Un seul modèle va retenir notre attention : c'est le **modèle relationnel**.

Remarque. Quand le SGBD implémente un modèle relationnel, on parle de SGBD**R**.

Parmi les avantages des SGBDR, on peut citer :

- la simplicité du modèle (à découvrir page suivante)
- l'indépendance physique / logique : l'organisation du stockage est invisible pour l'utilisateur
- la manipulation non-procédurale avec l'emploi du langage SQL (à découvrir dans un prochain chapitre)

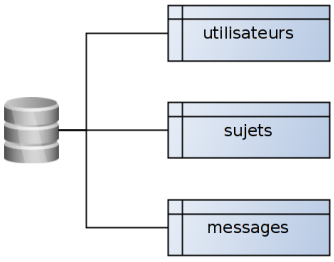
1.3 Base de données relationnelle

1.3.1 Modèle relationnel : table, attribut, enregistrement

Définition. Une **base de données relationnelle** est une base de données où l'information est organisée dans des tableaux à deux dimensions appelés des relations ou **tables**.

Considérons le problème suivant : vous devez développer un forum sur un site Web. Vous pourriez par exemple créer 3 tables :

- *utilisateurs* : les utilisateurs du site
- *sujets* : les sujets postés sur le forum
- *messages* : les messages écrits par les utilisateurs



Chaque table contient un certains nombre de colonnes, également appelées **attributs** ou **champs**, et un certain nombre de lignes ou **enregistrements**.

utilisateurs

ID	Nom	Prénom	Email	Mot de passe	Age
1	Orsoni	Françoise	françoise.orsoni-9@mail.fr	toto	74
2	Filippi	Sophie	sophie.filippi-16@supermail.com	R8dI$EOc1	41
3	Mondoloni	Andria	andria.mondoloni@mail.com	12345	25

sujets

ID	ID Utilisateur	Intitulé	Date d'ajout	Vues	Résolu
1	1	Demande d'aide	2020-06-01 11:34:51	57	1
2	3	Super ce forum !	2020-06-03 21:51:02	32	0
3	3	Votre avis sur la nouvelle RTX 3080	2020-06-04 09:23:36	129	0

messages

ID	ID Sujet	ID Utilisateur	Contenu	Date d'ajout
1	1	1	...	2020-06-01 11:34:51
2	1	2	...	2020-06-01 14:22:13
3	2	3	...	2020-06-03 21:51:02
4	1	1	...	2020-06-03 22:54:57
5	3	3	...	2020-06-04 09:23:36

Exemple. Notre table *utilisateurs* contient 6 colonnes. Chaque utilisateur est ainsi défini par 6 attributs : un identifiant *ID*, un nom, un prénom, un email, un mot de passe et un âge. Pourquoi 6 et pas 4 ou 10 ? Tout dépend du choix du programmeur !

1.3.2 Clés primaires et étrangères

Définition. Une **clé primaire** est un attribut ou groupe d'attributs dont les valeurs permettent d'identifier de manière unique les enregistrements d'une table.

Exemple. L'attribut *ID* de la table *utilisateurs* permet d'identifier un utilisateur de manière unique : on définit alors cet attribut comme clé primaire de la table.

▷ ***La contrainte d'unicité est imposée dans le modèle relationnel : on la respectera toujours !***

Utiliser un attribut *ID* est une pratique courante, car il n'est pas toujours possible de trouver un attribut qui soit unique pour chaque utilisateur.

Définition. Une **clé étrangère** est un attribut lié à la clé primaire d'une table.

Exemple. Chaque sujet est rattaché à un utilisateur unique (l'auteur du sujet), identifié par l'attribut *ID Utilisateur*, faisant le lien avec la table *utilisateurs*. On dit que cet attribut est une **clé étrangère**.

L'attribut *ID Utilisateur* de la table *sujets* est lié à l'attribut *ID* de la table *utilisateurs* : c'est une clé étrangère.

Clé Primaire

utilisateurs

ID	Nom	Prénom	Email	Mot de passe	Age
1	Orsoni	Françoise	françoise.orsoni-9@mail.fr	toto	74
2	Filippi	Sophie	sophie.filippi-16@supermail.com	R8dI$EOc1	41
3	Mondoloni	Andria	andria.mondoloni@mail.com	12345	25

sujets

ID	ID Utilisateur	Intitulé	Date d'ajout	Vues	Résolu
1	1	Demande d'aide	2020-06-01 11:34:51	57	1
2	3	Super ce forum !	2020-06-03 21:51:02	32	0
3	3	Votre avis sur la nouvelle RTX 3080	2020-06-04 09:23:36	129	0

Clé Étrangère

Question 01 Décrire les clés primaire et étrangères de la table *messages*.

1.3.3 Types de données

Chaque attribut possède un **type** différent : certains attributs sont des entiers (comme l'âge d'un utilisateur), d'autres des chaînes de caractères (comme l'intitulé d'un sujet ou le contenu d'un message), certains sont des dates (comme la date d'ajout d'un message) ou encore des booléens (comme le caractère *résolu* d'un sujet).

Les types de données dépendent du SGBD utilisé. Pour celui que nous utiliserons (*MySQL*), voici les principaux types utilisés :

Entiers			
Type	**Taille (octets)**	**Min**	**Max**
`TINYINT`	1	-128	127
`SMALLINT`	2	-32768	32767
`MEDIUMINT`	3	-8388608	8388607
`INT`	4	-2147483648	2147483647
`BIGINT`	8	-2^{63}	$2^{63}-1$

Remarque. Il est possible de spécifier un entier non signé avec l'attribut `UNSIGNED`.
Par exemple, le type « `UNSIGNED INT` » permet de représenter des entiers de 0 à $2^{32}-1 = 4\,294\,967\,295$.

Flottants	
Type	**Taille (octets)**
`FLOAT`	4
`DOUBLE`	8

Chaînes de caractères		
Type	**Taille (octets)**	**Commentaire**
`VARCHAR`	L + 1	De 0 à 255 caractères
`TEXT`	L + 2	De 0 à 65535 caractères

L : taille de la chaîne de caractères

Dates		
Type	**Taille (octets)**	**Format**
`DATE`	3	AAAA-MM-JJ
`DATETIME`	8	AAAA-MM-JJ HH :MM :SS

☞ *La valeur d'un attribut peut être nulle, sa valeur est alors* `NULL`*. Par exemple, si un utilisateur ne renseigne pas son âge, on peut utiliser la valeur* `NULL` *pour spécifier que cette valeur n'est pas renseignée.*

Pour en savoir plus :

`https://openclassrooms.com/fr/courses/1959476-administrez-vos-bases-de-donnees-avec-mysql/1960456-distinguez-les-differents-types-de-donnees`

Question 02 Proposer un type pour chacun des attributs des tables *utilisateurs*, *sujets* et *messages*.

1.3.4 Contraintes d'intégrité

Un des rôles les plus importants d'un SGBDR est la vérification des **contraintes d'intégrité**. Celles-ci sont vérifiées automatiquement à chaque opération de mise à jour de la base de données. Parmi ces contraintes, on trouve notamment :

- les contraintes de type : les valeurs d'un attribut doivent être conformes au type spécifié
- les contraintes de clés primaires : les valeurs des clés primaires doivent être **uniques** et **non nulles**
- les contraintes de clés étrangère : les valeurs des attributs clés étrangères doivent être incluses dans les valeurs des clés primaires associées
- des contraintes simples comme `NOT NULL` (valeur non nulle) ou `UNIQUE` (valeur unique)

Ce sont les contraintes d'intégrité qui font toute la force d'un SGBD.

Par exemple, si une contrainte `UNIQUE` est spécifiée sur l'attribut *Email* dans la table *utilisateurs*, il sera impossible d'avoir deux enregistrements avec la même adresse email. En cas d'ajout d'une adresse email déjà existante, le SGBD renverra une erreur mais ne mettra pas à jour la base de données.

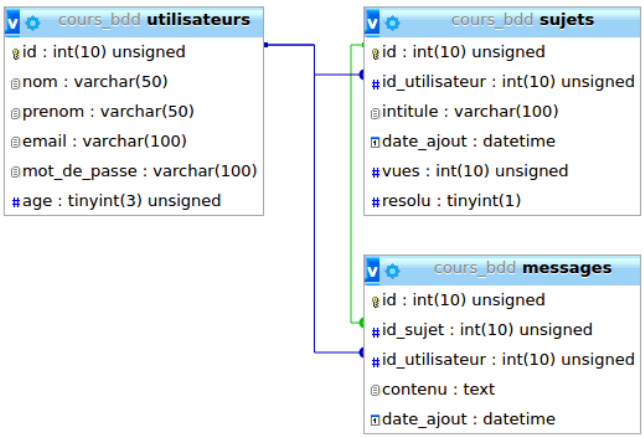
Le SGBD assure ainsi le maintient de l'intégrité de la base de données lors des opérations de mise à jour.

1.3.5 Schéma relationnel

Un schéma relationnel est un ensemble de schémas de tables définis chacun par :

- un nom
- une liste d'attributs (type et nom)
- une clé primaire
- des éventuelles clés étrangères (+ liens avec les clés primaires associées)

Voici par exemple le schéma relationnel de notre base de données précédente :



1.4 Opérations

Stocker des données, c'est bien, mais savoir les manipuler c'est encore mieux.
Voici quelques exemple d'opérations permises par un SGBD relationnel. La liste n'est pas exhaustive, et nous nous contenterons d'exemples simples dans cette partie théorique.
Dans cette partie, on considère les tables T_1 et T_2 suivantes :

T_1 : **utilisateurs**

ID	ID Ville	Nom	Prénom	Genre
1	1	Lucchini	Jules	H
2	5	Romani	François	H
3	3	Pietri	Élodie	F
4	2	Alfonsi	Saveria	F

T_2 : **villes**

ID	Nom Ville
1	Ajaccio
2	Bastia
3	Corte
4	Porto-Vecchio

☞ *Lorsque l'on réalise une opération sur une table ou plusieurs tables, on dit que l'on effectue une* ***requête***

1.4.1 Sélection

Définition. On considère une table T.
La *sélection* est l'opération qui consiste à ne retenir que les enregistrements de T qui vérifient une condition donnée.

Exemple. Résultat de la requête : « Sélection dans T_1 si genre = F »

ID	ID Ville	Nom	Prénom	Genre
3	3	Pietri	Élodie	F
4	2	Alfonsi	Saveria	F

☞ *Les résultats d'une requête ne sont pas donnés dans un ordre précis. Ils sont souvent triés selon la clé primaire, mais ce n'est pas toujours le cas*

1.4.2 Projection

Définition. On considère une table T.
La *projection* est l'opération qui consiste à ne retenir que certains attributs de T.

Exemple. Résultat de la requête : « Projection de T_1 sur ID et Nom »

ID	Nom
1	Lucchini
2	Romani
3	Pietri
4	Alfonsi

1.4.3 Jointure

Une jointure permet de créer une liaison entre deux ou plusieurs tables afin de récupérer un résultat bien précis. Le résultat est en quelque sorte une fusion virtuelle de plusieurs tables. Chaque enregistrement de chaque table est retourné dans le résultat en fonction de la jointure utilisée.
Il existe en effet plusieurs types de jointures.

Définition (Jointure interne). Soient T_1 et T_2 deux tables.
La **jointure interne** de ces deux tables sur un attribut A est une table formée en ne prenant que les enregistrements de T_1 et T_2 ayant une valeur de A commune.

Exemple. Jointure interne de T_1 sur T_2 sur l'attribut *ID Ville* :

ID	ID Ville	Nom	Prénom	Genre	ID Ville	Nom Ville
1	1	Lucchini	Jules	H	1	Ajaccio
3	3	Pietri	Élodie	F	3	Corte
4	2	Alfonsi	Saveria	F	2	Bastia

On remarque que l'utilisateur *Romani François* a disparu : en effet, il n'existe pas d'*ID Ville* égal à 5 dans la table T_2.

Définition (Jointure externe gauche). Étant données deux tables T_1 et T_2, la **jointure externe gauche** de T_1 avec T_2 sur l'attribut A est une table réunissant :

- les enregistrements de T_1 et T_2 ayant une valeur de A commune (jointure interne)
- les enregistrements de T_1 (table de **gauche**) pour lesquels l'attribut A de T_1 ne correspond avec aucun atrtibut A de T_2

Exemple. Jointure externe gauche de T_1 sur T_2 sur l'attribut *ID Ville* :

ID	ID Ville	Nom	Prénom	Genre	ID Ville	Nom Ville
1	1	Lucchini	Jules	H	1	Ajaccio
2	5	Romani	François	H	*NULL*	*NULL*
3	3	Pietri	Élodie	F	3	Corte
4	2	Alfonsi	Saveria	F	2	Bastia

Cette fois-ci, tous les enregistrements de T_1 sont conservés, donc François Romani n'a pas disparu. Mais comme aucune correspondance n'est faite avec la table T_2, les valeurs de *ID Ville* et *Nom Ville* sont égaux à *NULL*.

Définition (**Jointure externe droite**)**.** Étant données deux tables T_1 et T_2, la **jointure externe droite** de T_1 avec T_2 sur l'attribut A est une table réunissant :

- les enregistrements de T_1 et T_2 ayant une valeur de A commune (jointure interne)
- les enregistrements de T_2 (table de **droite**) pour lesquels l'attribut A de T_2 ne correspond avec aucun atrtibut A de T_1

Exemple. Jointure externe droite de T_1 sur T_2 sur l'attribut *ID Ville* :

ID	ID Ville	Nom	Prénom	Genre	ID Ville	Nom Ville
1	1	Lucchini	Jules	H	1	Ajaccio
3	3	Pietri	Élodie	F	3	Corte
4	2	Alfonsi	Saveria	F	2	Bastia
NULL	*NULL*	*NULL*	*NULL*	*NULL*	4	Porto-Vecchio

Dans ce cas, François Romani n'est pas conservé (son *ID Ville* n'apparaît pas dans la table T_2) mais la ville de Porto-Vecchio est conservée, bien que personne n'habite à Porto-Vecchio.