SGBD. Conception

La quantité de données dont nous disposons est devenue gigantesque. Pour les utiliser, il faut savoir les stocker, les traiter, les mettre en relation. La rapidité de leur utilisation a changé de nombreuses activités comme les échanges boursiers, la gestion des salaires dans le secteur public, la réservation de voyage ou de spectacles, la traduction automatique.

I/ Vocabulaire des bases de données

1/ Limites d'une structure plate

L'exploitation de **fichiers .csv** est relativement aisée, pratique et simple à mettre en œuvre puisqu'il s'agit d'un fichier texte que l'on peut compléter à l'aide d'un tableur.
Seulement, lorsque les données et critères deviennent nombreux et imbriqués, les requêtes deviennent lourdes à gérer et peu efficaces.

On considère par exemple la représentation d'un jeu multi-joueurs dans lequel chaque joueur dispose d'un pseudonyme, appartient à un peuple et dispose d'alliance. Il dirige par ailleurs plusieurs villages possédant entre autres une population et des coordonnées.
On pourrait représenter cela en Python à l'aide d'un dictionnaire où les clés seraient les pseudonymes des joueurs et leurs valeurs associées sont une liste contenant leur peuple, leur alliance et une liste de villages codés eux même sous forme de liste.
Par exemple :

```
Jeu = {'Astérix': ['Gaulois', 'Formidables', [['Le Village', 57,
    (34, 22)], ['Autre Village', 123, (36, 23)]]],
    'Obélix': ['Gaulois', 'Formidables', [['Le Village', 161,
    (38, 28)]]],
    'César': ['Romain', 'Pax', [['Rome', 1668, (151, 12)]]]}
```

S'il est simple de déterminer l'alliance à laquelle il appartient ou la population totale de ses villages comme ici :

```
def alliance_joueur(jeu, j):
    return jeu[j][1]

def population_joueur(jeu, j):
    return sum([village[1] for village in jeu[j][2]])
```

Il est plus compliqué de déterminer par exemple la population totale des villages contrôlés par un peuple (Gaulois ou Romain ici) :

```
def population_peuple(jeu, p):
    s = 0
    for joueur in jeu.values():
        if joueur[0] == p:
            s += sum([village[1] for village in joueur[2]])
    return s
```

Dans cette représentation de données, dite plate, une appartenance est privilégiée par rapport aux autres (l'appartenance à une alliance, à un peuple par exemple).
Or, on peut être amené à effectuer des recherches indépendamment de tout lien, ce que ne permet que difficilement ce système. Un nouveau modèle s'avère donc nécessaire dans lequel les données sont représentées dans plusieurs tables liées entre elles et ce sans « hiérarchie » particulière : la **base de données**.

2/ Système de gestion de bases de données

Une **base de données** peut être caractérisée par **un ensemble de données reliées entre elles** en suivant **certaines règles**.
Un **SGBD** (**S**ystème de **G**estion de **B**ase de **D**onnées) est un **outil informatique** permettant **la sauvegarde, l'interrogation, la recherche** et la mise en forme de **données**.

Quelques exemples de SGBD : Oracle Database, 4D, Microsoft SQL Server, SQLite, MySQL. Les deux derniers sont des logiciels en open-source (libre) et gratuits et sont par conséquent très utilisés, aussi bien par le grand public que par les professionnels. On utilisera cette année MySQL.

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

- **Assure** le **partage sécurisé des données** et leur **persistance**.
- **Gère** les **accès concurrents** (plusieurs clients utilisent la SGBD à partir de différents lieux).
- **Vérifie** qu'une **opération s'effectue entièrement ou pas du tout** (principe d'atomicité).
- **Protège** les **données** contre tout incident, notamment les **suppressions accidentelles**.
- **Optimise les performances** (temps de recherche).

Il existe plusieurs modèles de SGBD, les premiers étant les modèles *hiérarchiques* et *réseaux* (dépassés maintenant), le modèle *relationnel*, le plus répandu et celui que l'on étudiera et le modèle *objet* qui, comme son l'indique, encapsule les données dans des classes.

En savoir plus sur les modèles d'antan : https://cours.ebsi.umontreal.ca/sci6306/co/sgbd_modeles_historique.html
En savoir plus sur le modèle objet : https://pageperso.lis-lab.fr/bernard.espinasse/Supports/BD/SGBDO-4p.pdf
(complexe à comprendre mais donne quelques exemples).

3/ Schémas relationnels

Un peu de vocabulaire (à savoir)

- Un **attribut** est un couple (nom, domaine) où le nom (représentant un titre de colonne) prend des valeurs autorisées (domaine).
- Un **domaine** définit à la fois le type de valeurs (entier, date, chaîne de caractère (char) etc.) et éventuellement leur plage (nombre positif par exemple).
- On appelle **schéma** d'une relation un ensemble ordonné d'attributs distincts.
- Une **relation** ou **table** est un ensemble fini de tuples composés d'une valeur pour chaque attribut d'un schéma.
- Les éléments d'une relation sont appelés **enregistrements**. Leur nombre est le **cardinal** de la relation.

Voici un exemple de représentation de quelques données d'élèves de Terminale d'une relation nommée ELEVES

Nom	Prénom	Date_de_naissance	Spécialité_1	Spécialité_2
Garcin	Mélissande	13/01/2001	Maths	SPC
Berteix	Paul	11/08/2001	Maths	NSI
Laplaud	Alice	12/01/2002	NSI	SPC

Le schéma de cette relation est ELEVES((Nom, *char*), (Prénom, *char*), (Date_de_naissance, *date*), (Spécialité_1, *char*), (Spécialité_2, *char*).
La relation compte 3 enregistrements, son cardinal est donc de 3.

A savoir : lorsqu'une base de données contient plusieurs tables, l'ensemble des schémas de ces relations est le schéma relationnel de la base de données.

II/ Le modèle relationnel

1/ Historique

Le modèle relationnel a été imaginé dans les **années 1970** par l'informaticien britannique **Edgar Codd**. Avant cela, les modèles des bases de données (*hiérarchiques* et réseaux en particulier) ne permettaient de décrire de façon satisfaisante les relations entre deux données. Dix ans de recherches ont été nécessaires avant d'y parvenir et de publier l'ouvrage *« un modèle de données relationnel pour de grandes banques de données partagées » (traduction du titre en anglais 😊).*

En savoir plus sur Edgar Codd : https://www.techno-science.net/definition/7438.html

Les bases de données gérant actuellement un grand nombre de données (de l'ordre du *pétaoctet*, 10^{15} octets), il est essentiel d'identifier rapidement une information. Les notions de **clés primaires** et de **clés étrangères** vont le permettre, sur une ou plusieurs **tables**.

2/ Clés primaires

Le rôle d'une **clé primaire** est de permettre d'**identifier un enregistrement unique**. **Cela signifie qu'il ne peut y avoir deux enregistrements identiques car ils posséderaient la même clé primaire.**

Dans l'exemple de représentation donné précédemment, il ne peut y avoir de clé primaire !

Nom	Prénom	Date_de_naissance	Spécialité_1	Spécialité_2
Garcin	Mélissande	13/01/2001	Maths	SPC
Berteix	Paul	11/08/2001	Maths	NSI
Laplaud	Alice	12/01/2002	NSI	SPC

En effet, il peut y avoir deux prénoms, noms, date de naissance et spécialités identiques. Cette table est donc incomplète pour être relationnelle.

Deux solutions peuvent être proposées :

- Utiliser une clé primaire **composée de plusieurs attributs** (on peut proposer *Nom* + *Prénom* + *Date_de_naissance*). On peut être raisonnablement certain de son unicité au sein d'un établissement scolaire.
- Procéder à une clé primaire **gérée automatiquement par auto-incrémentation** d'un nombre entier naturel.

Exemple de clé auto-incrémentée

Id_eleve	Nom	Prénom	Date_de_naissance	Spécialité_1	Spécialité_2
1	Garcin	Mélissande	13/01/2001	Maths	SPC
4	Berteix	Paul	11/08/2001	Maths	NSI
2	Laplaud	Alice	12/01/2002	NSI	SPC

Important : rien n'indique que les enregistrements soient ordonnés selon les clés auto-incrémentées dans une table (suppression d'enregistrements etc.).

3/ Clés étrangères

Exemple : on souhaite créer une table de gestion de clients dans un magasin. Les informations demandées sont le prénom, nom et l'adresse email.
On peut proposer la table à droite.

id_client	prénom	nom	email
1	Jean	Dupond	jdupond@gmail.com
2	Marie	Talon	mtalon@yahoo.fr
3	Tom	Vigneux	tvigneux@orange.fr
4	Lucie	Métayer	lmetayer@wanadoo.fr

id_client représente ici la **clé primaire** et permet ainsi d'identifier aisément le client cherché.

On souhaite désormais gérer les achats que les clients dans cette même table. Voici un exemple que l'on pourrait donner :

numéro	id_client	prénom	nom	email	produit	prix unitaire	quantité
1	1	Jean	Dupond	jdupond@gmail.com	Tubes de colle	2,20	5
5	1	Jean	Dupond	jdupond@gmail.com	Ciseaux	4,20	3
10	1	Jean	Dupond	jdupond@gmail.com	Feuilles A4	3,10	5
3	2	Marie	Talon	mtalon@yahoo.fr	Ciseaux	4,20	10
4	2	Marie	Talon	mtalon@yahoo.fr	Feuilles calque	3,80	2
11	3	Tom	Vigneux	tvigneux@orange.fr			
13	4	Lucie	Métayer	lmetayer@wanadoo.fr	Tubes de colle	2,20	1
22	4	Lucie	Métayer	lmetayer@wanadoo.fr	Ciseaux	4,20	7
12	4	Lucie	Métayer	lmetayer@wanadoo.fr	Feuilles calque	3,80	2
9	4	Lucie	Métayer	lmetayer@wanadoo.fr	Copies doubles	1,90	3

Ici, l'attribut *numéro* devient la clé primaire, en effet, un client peut commander plusieurs produits (du moins, le magasin l'espère 😊) et à ce titre, son identifiant est susceptible d'être répété plusieurs fois.

Cette table, tout à fait utilisable, comporte de nombreux défauts :

- Il y a de **nombreuses répétitions** ce qui augmente le risque d'erreurs lors de la saisie. Cela augmente aussi le temps de recherche d'informations.
- Elle est **peu lisible** du fait d'un nombre d'attributs importants et des répétitions.
- Il y a un enregistrement incomplet (avec des « vides »). On peut se poser de l'utilité d'entrer les coordonnées d'un client qui n'a rien acheté dans cette table.

La **clé étrangère** va permettra de gommer tous ces défauts et d'assurer ainsi une meilleure protection de la future base de données.
On va séparer les clients de leurs commandes. Pour cela, on va donc créer une table *Commande*s.

Table « Clients »

id_client	prénom	nom	email
1	Jean	Dupond	jdupond@gmail.com
2	Marie	Talon	mtalon@yahoo.fr
3	Tom	Vigneux	tvigneux@orange.fr
4	Lucie	Métayer	lmetayer@wanadoo.fr

Table « Commandes »

numéro	client	produit	prix unitaire	quantité
1	1	Tubes de colle	2,20	5
5	4	Ciseaux	4,20	3
10	1	Feuilles A4	3,10	5
3	2	Ciseaux	4,20	10
4	2	Feuilles calque	3,80	2
13	4	Tubes de colle	2,20	1
22	1	Ciseaux	4,20	7
12	4	Feuilles calque	3,80	2
9	4	Copies doubles	1,90	3

Il y a donc désormais deux tables, qui vont être **reliées** par **l'attribut *id_client* pour la table *Clients*** et **l'attribut client pour la table *Commandes***. Ce dernier est appelé **« clé étrangère »**.

Cette **clé étrangère** permet donc plusieurs choses :

- D'éviter les **erreurs de saisies** lors de passage de commande d'un commande : si le numéro de client n'existe pas, la base de données le signalera et un client non existant ne pourra pas passer de commandes. De même, un client ayant commandé un produit ne pourra pas être supprimé de la table *Client*s. On appelle cela une **contrainte d'intégrité**.
- D'augmenter la **sécurité de la gestion de la base de données** : on peut donner l'autorisation à certaines personnes de ne voir que la table *Commandes* et masquer ainsi des données personnelles des clients.
- D'éviter les **répétitions** : gain de temps pour l'opérateur de saisie et la recherche d'information.

A savoir (à comprendre surtout, cela doit devenir logique) :

- **On aurait tout à fait pu donner le même nom à la clé primaire qu'à la clé étrangère.**
- **La clé étrangère ne peut exister que si elle est reliée à une clé primaire.**
- **Une clé primaire ne peut être modifiée si elle est reliée à une clé étrangère.**
- **Une clé étrangère peut aussi être composée de plusieurs attributs (c'est relativement rare).**
- **Une clé étrangère peut apparaître plusieurs fois dans une table (c'est le but même, sinon pourquoi en créer une ?).**
- **Une table peut comporter plusieurs clés étrangères.**

4/ Contraintes d'intégrité

Une **contrainte d'intégrité** est une règle qui définit la **cohérence** d'une **donnée** ou d'un **ensemble de données** dans une base de données.

A savoir (à comprendre surtout, cela doit devenir logique) :

- **Le type (et éventuellement la plage de valeurs) de données que l'on cherche à stocker définit une contrainte de domaine. Cela est intégré dans le modèle du schéma relationnel.**
- **Chaque enregistrement doit pouvoir être identifié par une clé primaire unique et non nulle. Il s'agit d'une contrainte de relation.**

Cette ensemble de règles (avec celle du 2/) est au cœur même des bases de données et confère le **caractère relationnel** au modèle étudié.

III/ Conception d'une base de données

1/ Introduction

Le respect des contraintes d'intégrité et de normalisation permet d'éviter la survenue d'anomalies dans les bases de données. Le **modèle entité-association** aboutit à un **modèle conceptuel de données** (MCD) qui conduira au **schéma relationnel** et enfin à l'élaboration de la base de données.

Lorsque les bases de données n'ont pas été suffisamment travaillées, des anomalies peuvent apparaître comme des tables comportant des **champs** (attributs) **sans lien logique** ou des **redondances** d'information par exemple.
On peut prendre l'exemple de la table naïve du II/2/.

Cela a évidemment un coût pour l'exploitation de la base de données (coût en mémoire, performances moindres) et donc une répercussion négative pour l'entreprise.
Comme disent les gobelins de *Ratchet* de *World of Warcraft*, « le temps, c'est de l'argent » 😊.

2/ Le modèle entité-association

Pour construire une base de données convenable, il faut dans un premier temps s'appuyer sur un **cahier des charges** très précis :

- Quels sont les *acteurs* (appelés **entités**) ?
- Quelles **données** pour **chacun d'entre eux** ?
- Quelles *liaisons* (appelées **associations**) **entre eux** ? Une entité peut-elle être liée plusieurs fois avec d'autres ou une seule fois ?

A partir de ce cahier des charges, on peut décrire précisément les entités nécessaires ainsi que leurs associations.

Exemple : on considère l'étude simplifiée de la gestion d'un parc immobilier :

- Des propriétaires possèdent, éventuellement collectivement, un ou plusieurs appartements.
- Un locataire occupe un seul logement (logique en somme). Pas de colocation autorisée.
- Les données du propriétaire sont *nom* et *prénom* ; celle de l'appartement *l'adresse* ; celles du locataire *nom* et *prénom*.

Trois **entités** se présentent ici : les *propriétaires*, les *locataires* et les *appartements*.
Deux **liens** existent : un locataire *habite* un appartement et un propriétaire *possède* un ou plusieurs appartements.

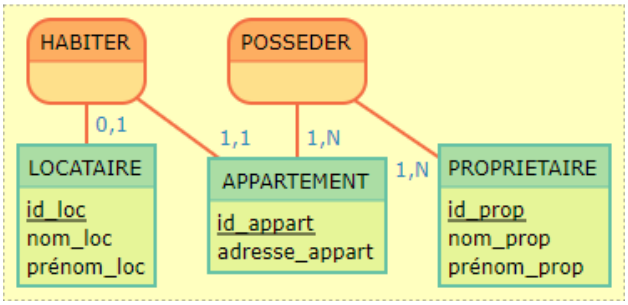
Voici le **modèle entité-association** correspondant :

ASSOCIATIONS
 # Dans un appartement, il y un locataire au maximum (01 signifie de « 0 à 1 »)
 # Un locataire ne peut habiter qu'un appartement (11 signifie de « 1 à 1 » donc 1)
HABITER, 01 LOCATAIRE, 11 APPARTEMENT

 # Un propriétaire possède au moins un appartement (1N signifie de 1 à N, nombre entier quelconque supérieur ou égal à 2)
 # Un appartement est possédé par au moins un propriétaire (1N signifie de 1 à N, nombre entier quelconque supérieur ou égal à 2)
POSSEDER, 1N PROPRIETAIRE, 1N APPARTEMENT

ENTITES
 # Les id_xxx sont des clés primaires par auto-incrémentation
LOCATAIRE : id_loc, nom_loc , prénom_loc
APPARTEMENT : id_appart, adresse_appart
PROPRIETAIRE : id_prop, nom_prop, prénom_prop

On en déduit le **modèle conceptuel des données (MCD)** sous forme de schéma :



Les **clés primaires** sont ici soulignées et sont ici des index auto-incrémentés.

On peut lire ce schéma ainsi :

- « Un locataire ne peut **habiter** qu'un appartement »
- « Un appartement **est habité** au maximum **par** un locataire »
- « Un propriétaire **possède** au moins un appartement »
- « Un appartement **est possédé par** au moins un propriétaire »

On peut observer deux types d'associations :

- Une **relation** dite **binaire fonctionnelle** lorsque deux entités sont reliées par au plus une occurrence (sans « N » sur le schéma précédent).
- Dans les autres cas, il s'agit d'une **association non fonctionnelle**.

3/ D'un MCD au schéma relationnel

D'un **MCD**, on peut en déduire le **schéma relationnel** en suivant les règles suivantes :

- Toute **entité** donne lieu à une **table** dont la clé primaire est son identifiant.
- Toute **association binaire fonctionnelle** devient une **clé étrangère** dans la table représentant l'entité d'origine (ici, l'association « HABITER » devient une clé étrangère dans la table APPARTEMENT car un seul locataire peut l'occuper au maximum).
- Toute **association non fonctionnelle** devient une **table** dont la **clé primaire** est formée de **l'ensemble de clés primaires des tables qu'elle relie** (ici, l'association « POSSEDER » qui relie les entités « PROPRIETAIRES » et « APPARTEMENT » devient une table avec pour clé primaire l'ensemble des clés *id_prop* et *id_appart*).
  Remarque : on peut tout à fait créer un index auto-incrémenté pour éviter cet ensemble de clés.

POSSEDER (*id_prop*, *id_appart*)
LOCATAIRE (id_loc, nom_loc, prénom_loc)
APPARTEMENT (id_appart, adresse_appart, *id_loc*)
PROPRIETAIRE (id_prop, nom_prop, prénom_prop)

Les contraintes en résumé

Contrainte de domaine

Les données ont un **type** et éventuellement une plage de valeurs.

Contrainte de relation

Chaque **enregistrement** d'une table doit pouvoir être identifié par une **clé primaire unique et non nulle**.

Normalisation d'une base de données

Les **attributs** doivent être **atomiques**, c'est-à-dire ne pas désigner plusieurs données en même temps. Ils possèdent en outre une valeur unique.

Contrainte d'intégrité référentielle

- Une **clé étrangère** ne peut **exister** que si elle est **en relation avec une clé primaire** d'une autre relation.
- Un **enregistrement de la table** primaire **ne peut être effacé** s'il possède des **liens** vers d'autres tables (via des clés étrangères).
- Une **clé primaire ne peut être changée** si elle possède des **liens** vers d'autres tables (clés étrangères).