|  |
| --- |
| Capacités attendues |
| * Identifier les concepts définissant le modèle relationnel. * Savoir distinguer la structure d’une base de données de son contenu. * Repérer des anomalies dans le schéma d’une base de données. * Identifier les services rendus par un système de gestion de bases de données relationnelles : persistance des données, gestion des accès concurrents, efficacité de traitement des requêtes, sécurisation des accès. |

# Limites des structures plates

1. **Exemple : représentation d’un jeu multi-joueurs**

Dans un jeu multi-joueurs, chaque joueur possède un pseudonyme, appartient à un peuple et à une alliance. Il dirige un ou plusieurs villages, qui ont chacun un nom qu’il choisit ; ces villages possèdent, entre autres choses, une certaine population et ont des coordonnées spatiales.

Une représentation en Python pourrait consister en un dictionnaire où les clés sont les joueurs et leurs valeurs associées sont une liste contenant leur peuple, leur alliance et une liste de villages, ceux-ci étant eux-mêmes codés par une liste. Par exemple :

Jeu = {‘Asterix’ : [‘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)]]]}

1. **Les limites de la représentation**

Le choix de cette représentation n’est pas anodin : il est facile de déterminer l’alliance à laquelle appartient un joueur ou la population totale des villages qu’il contrôle :

def alliance\_joueur(jeu, j) :

return jeu[j][1]

def population\_joueur(jeu, j) :

return sum([village[1] for village in jeu[j][2]])

Cependant, il est plus malaisé d’extraire certaines informations comme la population totale des villages contrôlés par peuple :

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 (l’appartenance à une alliance, à un peuple, à un joueur, etc…). Or on peut être amené à effectuer des recherches indépendamment de tout lien. Un nouveau modèle s’avère donc nécessaire dans lequel les données sont représentées dans plusieurs tables, sans hiérarchie particulière, et liées entre elles.

# Systèmes de Gestion de Base de Données (SGBD)

1. **Introduction**

**Définition n°1 :**

Un *Système de Gestion de Base de Données (SGBD)* est un outil informatique permettant la sauvegarde, l’interrogation, la recherche et la mise en forme de données.

Un SGBD est donc un ensemble de logiciels systèmes qui permettent aux programmateurs d’insérer, de modifier et de rechercher des données spécifiques dans une grande masse d’informations partagées par plusieurs utilisateurs. Ces informations sont généralement enregistrées sur des disques magnétiques (par exemple, des serveurs). Un SGBD donne l’impression à chaque utilisateur qu’il est le seul à travailler avec les données.

Oracle Database, AD, Microsoft SQLServer, SQLite ou MySQLsont des exemples de SGBD très répandus. SQLite ou MySQL sont des logiciels libres (open source), et sont par conséquent très utilisés, aussi bien par le grand public que les professionnels.

1. **Fonctions d’un SGBD**

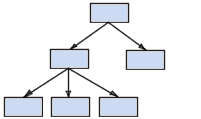
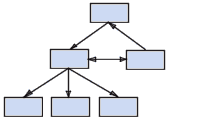
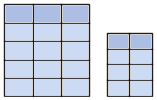
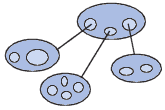
En plus des fonctions primaires citées dans la définition précédente, un SGBD :

* assure le partage des données ;
* vérifie qu’une opération s’effectue entièrement ou pas du tout : c’est ce que l’on nomme l’atomicité ;
* protège les données contre tout incident (détérioration, suppression accidentelle, etc…) ;
* optimise les performances (temps de recherche minimisé par exemple).

Les SGBD permettent la description des données (définition des types par des noms, formats…) de manière séparée de leur utilisation (mise à jour et recherche). Ils permettent aussi de retrouver les caractéristiques d’un type de données à partir de son nom (par exemple, comment est décrit un produit).

1. **Les différents modèles de SGBD**

Il existe cinq modèles de SGBD :

* **le modèle hiérarchique** : les données sont classées hiérarchiquement, selon une arborescence descendante. Ce modèle utilise des pointeurs entre les différents enregistrements. Il s'agit du premier modèle de SGBD  
  
* **le modèle réseau** : comme le modèle hiérarchique ce modèle utilise des pointeurs vers des enregistrements. Toutefois la structure n'est plus forcément arborescente dans le sens descendant  
  
* **le modèle relationnel** (SGBDR, Système de gestion de bases de données relationnelles) : les données sont enregistrées dans des tableaux à deux dimensions (lignes et colonnes). La manipulation de ces données se fait selon la théorie mathématique des relations  
  
* **le modèle déductif :** les données sont représentées sous forme de table, mais leur manipulation se fait par calcul de prédicats
* le modèle objet (SGBDO, Système de gestion de bases de données objet) : les données sont stockées sous forme d'objets, c'est-à-dire de structures appelées classes présentant des données membres. Les champs sont des instances de ces classes  
  

Depuis la fin des années 1990, le modèle relationnel est le plus répandu (environ trois quarts des bases de données utilisent ce modèle). Dans la suite du cours, nous ne nous intéresserons qu’à ce modèle.

# Le modèle relationnel

1. **Historique**

Le modèle relationnel a été imaginé dans les années 1970 par l’informaticien britannique Edgar Frank Codd. Avant cela, les modèles de bases de données ne permettaient pas de décrire de façon satisfaisante les relations entre deux données à l’aide de pointeurs logiques. Dix années de recherches ont été nécessaires avant de publier l’article « A Relational Model of Data for Large Shared Data Banks » (un modèle de données relationnel pour de grandes banques de données partagées), CCAM 13, No. 6, June 1970.

1. **Représentations**

Comme nous l’avons dit dans la section précédente, le modèle relationnel consiste à représenter les données dans des tableaux, que l’on appelle tables. Prenons l’exemple d’un site internet, qui demande à ses utilisateurs trois données afin de pouvoir leur créer un compte : nom, prénom et email. On peut alors imaginer une table user regroupant les données des utilisateurs dans trois colonnes : surname, name et email.

|  |  |  |
| --- | --- | --- |
| user | | |
| surname | name | email |
| Dupont | Jean | [Jean.dupont@free.fr](mailto:Jean.dupont@free.fr) |
| Durant | Anne | Anne.dur@orange.fr |
|  |  |  |

1. **Définitions**

**Définitions n°2**

Dans une table, chaque ligne est un *enregistrement*.

Les *attributs* d’une table sont les noms de ses colonnes.

Le *degré* d’une table est le nombre de ses attributs.

Le *domaine* d’un attribut est son type : entier, flottant, chaîne de caractères, booléen…

La *clé primaire* d’une table est l’attribut qui permet d’identifier de manière unique (sans aucun risque de doublon) un enregistrement de cette table.

Une *clé étrangère* est un attribut d’une table qui fait référence à la clé primaire d’une autre table. Une clé étrangère permet de mettre en relation un enregistrement d’une table (appelée *table fille*) qui la contient avec un enregistrement de la table référencée (appelée *table parent*).

**Exemple**

Reprenons la table précédente et ajoutons-y une clé primaire id :

|  |  |  |  |
| --- | --- | --- | --- |
| user | | | |
| id | surname | name | email |
| 1 | Dupont | Jean | [Jean.dupont@free.fr](mailto:Jean.dupont@free.fr) |
| 2 | Durant | Anne | Anne.dur@orange.fr |
|  |  |  |  |

Les attributs de cette table sont donc : *id, surname, name et email*.

Le degré de la table est donc égal à 4 car il y a quatre attributs.

Le domaine de l’attribut id est : entier, et celui des autres attributs est : chaîne de caractères.

Imaginons maintenant que ce site internet propose plusieurs abonnements : « Free », « Basic » et « Premium ». On peut alors créer une seconde table nommée « abonnement » représentant les abonnements :

|  |  |
| --- | --- |
| abonnement | |
| id\_abo | type |
| 1 | Free |
| 2 | Basic |
| 3 | Premium |

On peut alors mettre en relations les tables « user » et « abonnement » en insérant une clé étrangère dans la première table :

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| user | | | | |
| id | surname | name | email | id\_abo |
| 1 | Dupont | Jean | [Jean.dupont@free.fr](mailto:Jean.dupont@free.fr) | 3 |
| 2 | Durant | Anne | Anne.dur@orange.fr | 1 |
|  |  |  |  |  |

On met ici en relation les deux tables en considérant que l’attribut *id\_abo* de la table « user » correspond à l’attribut *id\_abo* de la table « abonnement ».

Ainsi, si l’on regarde le premier enregistrement de la table « user », on peut voir que Jean Dupont a souscrit à un abonnement Premium, c’est-à-dire à l’enregistrement de la table « abonnement » dont l’attribut *id\_abo* est 3.

Application

1. **Schématisation**

Pour schématiser la relation qu’il y a entre deux tables, on peut utiliser la représentation suivante :

|  |
| --- |
| **Table 3** |
| \*id\_table3  #id\_table2 |
| attr1\_table3  attr2\_table3 |

|  |
| --- |
| **Table 1** |
| \*id\_table1  #id\_table2 |
| attr1\_table1  attr2\_table1  attr3\_table1 |

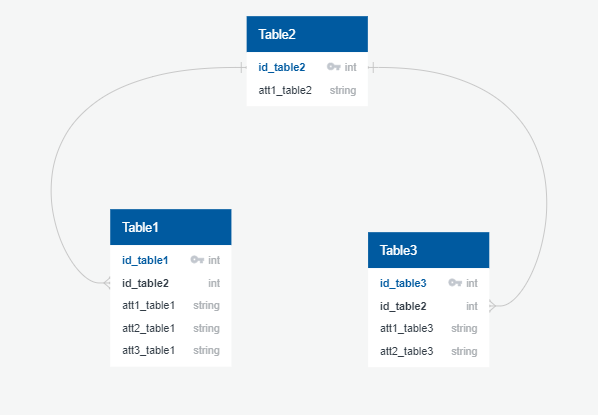
|  |
| --- |
| **Table 2** |
| \*id\_table2 |
| attr1\_table2 |

La relation entre deux tables est indiquée par un trait qui les lie.

Dans chacune des tables, on peut indiquer la présence d’une clé primaire en précédant l’attribut d’un astérisque (\*), et la présence d’une clé étrangère en précédant l’attribut d’un dièse (#).

On peut aussi, pour plus de clarté, séparer les clés des autres attributs.

Il existe également des logiciels et sites internets qui permettent de créer plus facilement ces schémas. Nous utiliserons dans ce cours le site : quickdatabasediagrams.com.



**Définition n°3**

Une *base de données* est un ensemble de tables, dont certaines peuvent être mises en relation.

Application

1. **Contraintes d’intégrité**

**Définition n°4**

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.

**Définition n°5**

Le type de données que l’on cherche à stocker définit une *contrainte de domaine*. Cela est intégré dans le modèle du schéma relationnel.

**Définition n°6**

Chaque enregistrement d’une relation doit pouvoir être identifié par une clé primaire, unique et non nulle. On parle dans ce cas de *contrainte de relation*.

Lorsque des relations sont liées, il est indispensable que les trois règles suivantes soient respectées :

* Une clé étrangère ne peut être une valeur qui n’est pas clé primaire de la table à laquelle elle se réfère.
* UN enregistrement de la table primaire ne peut être effacé s’il possède des enregistrements liés.
* La clé primaire ne peut être changée dans la table primaire si cet enregistrement possède des enregistrements liés.

Ces trois règles définissent la notion de *contrainte d’intégrité référentielle* d’une base de données. Cet ensemble de règles est au cœur même des bases de données et confère le caractère relationnel au modèle étudié.

Application

# Conception d’une base de données

1. **Exemple d’anomalie : redondance de données**

Lorsque les bases de données n’ont pas été suffisamment normalisées ou lorsqu’elles n’ont pas été conçues de sorte que les contraintes d’intégrité soient respectées après modification des relations, des *anomalies* peuvent apparaître, comme la *redondance de données*.

Considérons, par exemple, une base de données modélisant les produits achetés par une entreprise avec le schéma relationnel :

S = ((Produit, chaîne de caractère), (Fournisseur, chaîne de caractère), (Adresse, chaîne de caractère))

Un exemple de table pour ce schéma :

|  |  |  |
| --- | --- | --- |
| S | | |
| Produit | Fournisseur | Adresse |
| Tomate | Verger du sud | MIN Cavaillon |
| Ananas | Verger du sud | MIN Cavaillon |
| Origan | SA Epices | MIN Rungis |

L’adresse est redondante dans cette table, celle des Vergers du Sud apparaissant plusieurs fois, ce qui a un coût d’espace et conduit à des performances moindres de la base de données.

La solution typique consiste à séparer cette table en deux :

Fabricants = ((Fournisseur, chaîne de caractère), (Adresse, chaîne de caractère))

Produits = ((Produit, chaîne de caractère), (Fournisseur, chaîne de caractère))

|  |  |
| --- | --- |
| Fabricants | |
| Fournisseur | Adresse |
| Verger du sud | MIN Cavaillon |
| SA Epices | MIN Rungis |

|  |  |
| --- | --- |
| Produits | |
| Produit | Fournisseur |
| Tomate | Verger du sud |
| Ananas | Verger du sud |
| Origan | SA Epices |

Néanmoins cette solution n’est pas pleinement satisfaisante. Si la société Verger du sud change de nom ou est supprimée de la table « Fabricants », la table « Produits » aura une anomalie de mise à jour (ou de suppression), car les produits Tomate et Ananas ne seront plus référencés. Et si une nouvelle société prend le même nom qu’une déjà existante on aura une anomalie d’insertion.

1. **Elaboration d’un modèle conceptuel des données**

La première étape pour aboutir à un modèle permettant de stocker les données dans une base consiste à identifier les objets à représenter ainsi que leurs liens. Cela aboutit à un *modèle conceptuel des données (MCD)*, comme le *modèle entité-association* présenté ici.

**Définition n°7**

Une entité est une unité de base du système modélisé. Il peut s’agir d’individus, d’objets qui ont des propriétés communes.

**Définition n°8**

Une association représente un lien entre des entités. Elle peut être binaire et représenter un lien entre deux entités A et B, nommée par un verbe. Si chaque occurrence de l’entité A est liée à au plus une occurrence de l’entité B on parle de *relation binaire fonctionnelle*, sinon on parle *d’association non fonctionnelle.*

**Exemple :**

Prenons comme exemple l’étude simplifiée de la gestion d’un parc immobilier. Des propriétaires possèdent, éventuellement collectivement, un ou plusieurs appartements, un locataire occupant un appartement.

Trois entités se dégagent ici : les propriétaires, les locataires et les appartements. Il y a un lien représenté par le verbe « habiter » entre l’entité locataire et l’entité appartement. Ce lien est une relation binaire fonctionnelle dans la mesure où on admet qu’un locataire n’occupe qu’un seul appartement. Il y a un lien représenté par le verbe « posséder » entre l’entité propriétaire et l’entité appartement. Cette association n’est pas fonctionnelle car un propriétaire peut posséder plusieurs appartements et, réciproquement, ils peuvent être co-propriétaires.

1. **D’un MCD aux tables**

D’un MCD, on en déduit des tables liées entre elles en utilisant les règles suivantes :

* Toute entité donne lieu à une table dont la clé primaire est l’identifiant de l’entité.
* Toute association binaire fonctionnelle est implémentée par la présence d’une clé étrangère dans la table qui représente l’entité origine de la dépendance fonctionnelle.
* Toute association non fonctionnelle génère une table dont la clé primaire est l’ensemble des clés primaires des tables qu’elle relie.

**Exemple :**

Dans l’exemple de la gestion immobilière, on a trois entités évidentes donc trois tables : « propriétaires », « locataires » et « appartements ». Pour chaque table, on crée une clé primaire qui correspond à l’identifiant de l’entité et on ajoute les attributs utiles.

|  |
| --- |
| **appartements** |
| \*id\_appart |
| adresse |

|  |
| --- |
| **propriétaires** |
| \*id\_proprio |
| nom  prénom |

|  |
| --- |
| **locataires** |
| \*id\_locataire |
| nom  prénom |

Dans le MCD, il y avait une relation binaire fonctionnelle entre « locataires » et « appartements ». On ajoute donc dans chacune de ces tables une clé étrangère faisant référence à l’autre table :

|  |
| --- |
| **appartements** |
| \*id\_appart  #id\_locataire |
| adresse |

|  |
| --- |
| **locataires** |
| \*id\_locataire  #id\_appartement |
| nom  prénom |

|  |
| --- |
| **propriétaires** |
| \*id\_proprio |
| nom  prénom |

Il avait également une association non fonctionnelle entre « propriétaires » et « appartements » qui donne lieu à la création d’une nouvelle table « proprioAppart » contenant les clé primaires des tables « propriétaires » et « appartements » .

|  |
| --- |
| **appartements** |
| \*id\_appart  #id\_locataire |
| adresse |

|  |
| --- |
| **locataires** |
| \*id\_locataire  #id\_appartement |
| nom  prénom |

|  |
| --- |
| **propriétaires** |
| \*id\_proprio |
| nom  prénom |

|  |
| --- |
| proprioAppart |
| #id\_appart  #id\_proprio |