PRÉPAS SCIENCES

collection dirigée par Bertrand Hauchecorne

Informatique

Tronc commun

MP-PC-PSI-PT



Serge BAYS

Agrégé de mathématiques,
professeur d'informatique en classes préparatoires
au lycée Les Eucalyptus (Nice)

ellipses

COLLECTION
PRÉPAS SCIENCES

Retrouvez tous les titres de la collection et des extraits sur www.editions-ellipses.fr

Les notices culturelles « Un scientifique » et « Un peu d'histoire » des pages de titre des chapitres ont été rédigées par Bertrand Hauchecorne.

Les macros de cet ouvrage ont été réalisées par Nicolas Nguyen en LaTeX.

ISBN 9782340-066755
© Ellipses Édition Marketing S.A., 2022
8/10 rue la Quintinie 75015 Paris

DANGER LE PHOTOCOPILLAGE TUE LE LIVRE

Le Code de la propriété intellectuelle n'autorisant, aux termes de l'article L. 122-5.2° et 3°a), d'une part, que les « copies ou reproductions strictement réservées à l'usage privé du copiste et non destinées à une utilisation collective », et d'autre part, que les analyses et les courtes citations dans un but d'exemple et d'illustration, « toute représentation ou reproduction intégrale ou partielle faite sans le consentement de l'auteur ou de ses ayants droit ou ayants cause est illicite » (art. L. 122-4).
Cette représentation ou reproduction, par quelque procédé que ce soit constituerait une contrefaçon sanctionnée par les articles L. 335-2 et suivants du Code de la propriété intellectuelle.

www.editions-ellipses.fr

Avant-propos

Informatique commune en classes préparatoires

Le nouveau programme d'informatique commune en classes préparatoires scientifiques est entré en vigueur à la rentrée 2021. Il est composé de trois sections correspondant à trois semestres. Les deux premières sections sont étudiées en première année et la troisième section à laquelle est consacré ce livre est étudiée au premier semestre en deuxième année.

Le programme de deuxième année est divisé en trois parties comme cet ouvrage dans lequel chaque partie est composée de deux chapitres.
La première partie traite des bases de données relationnelles en présentant quelques concepts et des moyens de dialoguer avec une base de données à l'aide du langage SQL en se limitant à leur interrogation.
La deuxième partie présente les principes de la programmation dynamique et propose l'étude d'algorithmes classiques utilisant cette stratégie. Avant cela elle revient sur les dictionnaires qui, avec la notion de mémoïsation, permettent d'améliorer l'efficacité des algorithmes en leur ajoutant de la mémoire. Les dictionnaires ont été rencontrés en première année et sont ici étudiés pour eux-mêmes au niveau de leur utilisation et de leur implémentation.
La troisième partie aborde quelques notions présentes à la base de l'étude de l'intelligence artificielle et plus précisément des questions d'apprentissage avec deux algorithmes importants. D'autres algorithmes, ayant des applications dans divers domaines, sont introduits dans le cadre de l'étude des jeux à deux joueurs, jeux modélisés par des graphes représentés à l'aide de dictionnaires.
Chaque chapitre se termine par des exercices qui sont tous corrigés.

L'ensemble de l'ouvrage est strictement conforme au programme. En particulier, seuls les éléments du langage Python dont la connaissance est exigible, listés en annexe du programme, sont utilisés. Ces éléments, en nombre volontairement restreint, sont suffisants pour traiter toutes les questions d'algorithmique en Python. Ainsi, des simplifications proposées par le langage, qui n'apporteraient dans ce cadre aucun intérêt pédagogique, sont évitées.
De manière générale les programmes en Python sont écrits sans aucun commentaire pour un gain de place, ce qui n'est pas une pratique recommandable. Il est conseillé de les annoter en choisissant des commentaires judicieux.

L'ensemble a été relu avec soin. Il peut cependant subsister quelques erreurs dans le texte et les programmes. Si c'est le cas, j'espère que la lecture n'en sera pas trop perturbée.

Merci à Corinne Baud, Bertrand Hauchecorne et les éditions Ellipses pour leur confiance.

Serge Bays

Sommaire

- **Bases de données**
  1. Organisation des données 3
  2. Langage SQL 31

- **Dictionnaires et programmation dynamique**
  3. Dictionnaires 65
  4. Programmation dynamique 97

- **Algorithmique**
  5. Algorithmes et apprentissage 143
  6. Algorithmes et jeux 187

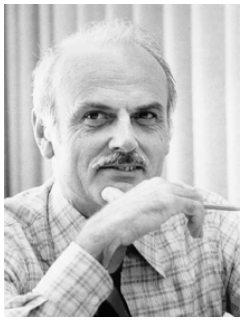
  Index 221

Bases de données

Chapitre 1

Organisation des données

UN SCIENTIFIQUE



Natif du sud de l'Angleterre, **Edgar Franck Codd** (1923-2003) suit des études de mathématiques et de chimie à Exeter puis à Oxford. Pendant la guerre, il sert comme pilote dans la Royal Air Force. Par la suite, il s'installe en Amérique du Nord (États-Unis et Canada) et se spécialise en informatique ; il s'intéresse plus particulièrement à la gestion des bases de données mais travaille aussi sur les automates cellulaires. Ses travaux furent récompensés par le prix Turing en 1981.

Un peu d'histoire

L'informatique s'est révélée d'une très grande efficacité pour manier des fichiers immenses ce qui a permis aux entreprises, comme à l'administration, une gestion rapide et efficace.

Pour ce faire, il a fallu concevoir des algorithmes très astucieux et d'une grande fiabilité.

Ceux qui trient les données sont fondamentaux car ils permettent l'accessibilité d'information dans des délais très brefs. Aussi, s'est-on attelé à concevoir ceux qui nécessitent le moins d'opérations. La scientifique américaine Betty Holberton (1917-2001) en a été une des pionnières.

Le mathématicien et informaticien américain Edgar Franck Codd (1923-2003) a conçu une théorie mathématique pour traiter de la manière la plus efficiente les bases de données, appelée modèle relationnel. Elle consiste en une modélisation des relations entre les données et en la manière de les ordonner.

Les bases de données permettent de stocker, organiser et analyser des données.

OBJECTIFS

- Comprendre les concepts du modèle relationnel :
  - acquérir les notions et le vocabulaire ;
  - être capable de construire un schéma relationnel simple ;
  - déterminer les éventuelles anomalies d'un schéma.
- Assimiler la notion de clé primaire :
  - connaître le rôle d'une clé primaire et mesurer son intérêt ;
  - comprendre la nécessité d'utiliser suivant les cas un seul attribut ou plusieurs.
- Aborder le modèle entités et associations :
  - connaître différentes représentations ;
  - être capable de décrire différents types d'associations ;
  - établir les liens avec les notions de clé primaire et de clé étrangère.

Cours

Introduction

Les données en général et la gestion de ces données sont omniprésentes dans de nombreuses activités. La quantité de données est gigantesque et se mesure en gigaoctet, téraoctet ou pétaoctet. L'utilisation de bases de données relationnelles permet le stockage de données ainsi que l'organisation des mises à jour et des consultations par un grand nombre d'utilisateurs.

Principes et architecture

- Les données sont stockées physiquement dans une machine.
- Le modèle relationnel représente la logique de l'organisation des données.
- L'utilisateur devant sa machine ne se préoccupe ni de l'aspect physique, ni de l'aspect logique.

Ces trois niveaux, physique, logique et externe, sont indépendants les uns des autres. Ainsi, une modification sur le schéma physique par exemple n'a aucune conséquence sur le schéma logique et les schémas externes.

Types de données

Les données traitées peuvent être de formats différents : des nombres, des textes, des images, des vidéos. Pour les stocker, il faut choisir comment les coder afin de les enregistrer sur un support numérique, donc définir un type de données, par exemple des types numériques, des types textes, des types dates. La manière de les stocker doit permettre d'accéder à des informations et d'effectuer des opérations simples comme un tri le plus rapidement possible.

Chaque système de gestion de bases de données, ou SGBD, présente quelques différences sur les types disponibles mais en propose suffisamment afin de pouvoir utiliser dans chaque cas le plus économique en besoin au niveau de la mémoire et limiter ainsi la taille de la base de données.

Les types numériques se décomposent en types entiers ou pas. Il y a en général plusieurs types prévus pour les entiers (selon la taille et le signe), mais pour la suite on se contente d'envisager un seul type *entier*.

Pour les nombres non entiers plusieurs types peuvent aussi être proposés mais on se limite dans la suite au type *flottant*.

Une représentation des nombres en machine a été étudiée en première année avec des mots de taille fixe (8, 16, 32, 64 bits). Les SGBD peuvent utiliser des représentations différentes pour certains types de nombres afin de pouvoir les stocker de manière exacte et effectuer des calculs exacts. Par exemple, pour un nombre décimal, on peut envisager de coder en binaire chaque chiffre de son écriture. Avec des mots de même taille pour chaque chiffre, nous avons besoin de quatre bits par chiffre. Par exemple, le nombre 68,3 est codé par 0110 1000 0011. L'exactitude nécessite plus de place en mémoire.

Pour les textes également on peut disposer de plusieurs types principalement liés à la taille maximale que l'on prévoit d'utiliser, mais dans la suite on se restreint simplement à un type *chaîne de caractères*.

Concernant les dates, il existe des types précis mais on se limite à des chaînes de caractères dans le format `aaa-mm-jj`, ce qui est suffisant pour pouvoir effectuer des comparaisons en utilisant l'ordre

lexicographique. On peut aussi utiliser des nombres entiers représentant le nombre de secondes écoulées depuis une date origine qui est classiquement le 1er janvier 1970.

■ Modèle relationnel

L'ensemble des informations à stocker doit être analysé afin d'être modélisé. Ensuite il s'agit de concevoir la base de données. D'une certaine manière, cela signifie implémenter le modèle.

Une différence essentielle avec l'utilisation d'un tableur pour stocker des données et effectuer des calculs avec ces données apparaît ici : la notion de modélisation. C'est cette modélisation qui permet d'avoir des liens entre les données, de les contrôler (sécurité, validité), de les traiter à l'aide d'un programme informatique, de les rendre accessibles simultanément à un grand nombre d'utilisateurs, de pouvoir gérer des volumes de données de très grande taille.

Pour modéliser des relations entre plusieurs informations et les ordonner entre elles, une réflexion préalable sur les traitements qu'on souhaite réaliser sur les données est essentielle et déterminante. Les principaux concepts qui interviennent sont la *relation*, le *domaine*, l'*identifiant* ou *clé primaire*, l'*identifiant externe* ou *clé étrangère*.

Remarque : en langage des bases de données, on parle de *table* plutôt que de relation et de clé primaire plutôt que d'identifiant. Dans la suite les termes sont interchangeables.

■ Un point d'histoire

Edgar Frank Codd surnommé Ted Codd, qui a reçu le prix Turing en 1981, travaillait au laboratoire IBM en 1970 et proposa un modèle logique pour gérer le partage de gros volumes de données. Le premier système vit le jour dix ans plus tard avec la société Oracle. Des notions mathématiques comme le langage ensembliste, les relations, et la logique sont à la base de son modèle.

■ Schéma relationnel

Les relations

Considérons dans un lycée des données concernant les élèves, les professeurs, les administratifs. On représente l'ensemble des données, comme le nom, le prénom, la date de naissance, le numéro de téléphone de chaque personne, dans un tableau à deux dimensions appelé *relation* ou *table*.

Une colonne du tableau est appelée *attribut* ou *champ*, par exemple la colonne nommée `nom`. L'ensemble des colonnes caractérisent la relation.

Les valeurs des attributs pour une personne particulière constituent une ligne qui est appelée *enregistrement* ou *n-uplet*, (*tuple* en anglais), ou *instance*. Chaque n-uplet est unique. Chaque valeur a un type, le même type pour toutes les valeurs contenues dans une colonne, au sens type de donnée en programmation (entier, texte, flottant, ...), avec des valeurs limites (par exemple les entiers naturels inférieurs à 256, les textes comportant au maximum 8 caractères, etc). On dit que les valeurs permises dans une colonne appartiennent à un *domaine*.

L'obligation d'appartenance d'une valeur à un domaine s'appelle une *contrainte d'intégrité*. Plus précisément, il s'agit ici d'une *intégrité de domaine*.

Le *produit cartésien* des domaines D_1, D_2, ..., D_n noté $D_1 \times D_2 \times \ldots \times D_n$ est l'ensemble des n-uplets $(v_1, v_2, \ldots, v_n)$ tels, pour tout i, v_i est un élément de D_i.

Une relation est donc un sous-ensemble du produit cartésien de domaines. Elle est caractérisée par un nom.

Par exemple, supposons donnés les domaines $D_1 = \{a, b, c\}$ et $D_2 = \{x, y\}$.

Alors : $D_1 \times D_2 = \{(a,x),(a,y),(b,x),(b,y),(c,x),(c,y)\}$.

D_1 \ D_2	x	y
a	(a,x)	(a,y)
b	(b,x)	(b,y)
c	(c,x)	(c,y)

Une relation est par exemple : $R = \{(a,x),(b,x),(c,y)\}$.

La relation R contient trois enregistrements ou trois lignes. On peut donner un nom à chacune des deux colonnes, la première qui contient des valeurs appartenant au domaine D_1, et la seconde qui contient des valeurs appartenant au domaine D_2

■ Notion de clé primaire

- Dans toute relation, un attribut, ou un groupe d'attributs, doit permettre d'identifier de manière unique un enregistrement. On l'appelle une *clé candidate*.
- S'il y a plusieurs clés candidates, on en privilégie une, nommée la *clé primaire*. Choisir une clé primaire est une contrainte d'intégrité imposée à la relation.
- Certaines relations peuvent contenir un attribut dont les valeurs permises sont uniquement celles prises par la clé primaire d'une autre relation. Cet attribut est alors appelé *clé étrangère*. Une clé étrangère est une *contrainte d'intégrité référentielle* (elle fait référence à un attribut d'une autre table). Elle relie deux tables de manière cohérente.

Les clés primaires et étrangères sont utilisées dans le modèle entité-association présenté dans la suite. Elles permettent de spécifier des contraintes dans les bases de données et d'effectuer des opérations de jointures entre les tables, opérations présentées dans le chapitre suivant.

La présence d'une clé primaire est capitale pour assurer que les n-uplets sont tous distincts. Une contrainte d'unicité est donc imposée : chaque valeur prise par l'attribut, ou le groupe d'attributs, sur lequel est posé une contrainte de clé primaire est unique.

De manière générale, une contrainte d'intégrité permet d'assurer une certaine cohérence dans les données. Il est par exemple impossible qu'une valeur de l'attribut clé étrangère d'une relation ne soit pas une valeur de l'attribut clé primaire de l'autre relation. Il est aussi impossible de supprimer une valeur de la clé primaire qui est une valeur de la clé étrangère.

■ Exemples

Considérons une base de données qui sert à stocker des informations dans un établissement scolaire. Pour représenter le concept d'élève, nous utilisons une relation `Élève`. Les attributs sont par exemple nommés `INE` (numéro d'identification), `Nom`, `Prénom`, `Âge`, `Sexe`, `Classe`. Chaque élève est représenté par un enregistrement. Une deuxième relation `Classe` représente le concept de classe, avec pour attributs `Nom`, `Effectif`, `Salle`. Il y a ensuite des liens entre l'ensemble des élèves et l'ensemble des classes. Un élève donné fait partie d'une classe particulière. Ce lien fonctionne dans les deux sens : une classe contient plusieurs élèves.

Les données peuvent être modélisées d'une autre manière. On ajoute une relation `Personne` pour représenter les personnes appartenant à la communauté scolaire. Si cette relation dispose d'attributs comme le nom, le prénom, etc., on peut alors supprimer ces attributs de la relation `Élève` et ne garder que le numéro INE et la classe. Pour des relations comme `Professeur`, `Administratif`,

`Service`, on peut remplacer l'attribut INE par un attribut ayant pour valeur le numéro de sécurité sociale par exemple.

Méthode

On définit un *schéma de relation*. Pour cela, on donne le nom de la relation, on précise le nom des attributs et pour chaque attribut le domaine de valeurs. Ce domaine est défini par un type et par des restrictions. Le domaine est une contrainte d'intégrité qui est garantie par le SGBD.

Préciser quelles sont les valeurs admises pour un attribut, le type de ces valeurs et les bornes, permet d'ajouter du sens. Par exemple, décider que le nom est une chaîne de caractères avec un nombre maximal de caractères ou que le sexe est un caractère "F" ou "M".

Un schéma de relation se présente sous cette forme :
`Relation (attribut 1, attribut 2, ..., attribut N)`
Clé primaire : `attribut 1`
Clé étrangère : `attribut N` en référence à la clé primaire d'une autre relation

Par exemple :
`Élève (INE, Nom, Prénom, Âge, Sexe, Classe)`
Les attributs `INE` et `Âge` sont de type entier, les autres de type chaîne de caractères.
Clé primaire : `INE`
Clé étrangère : `Classe` en référence à la clé primaire d'une autre relation.

Ce schéma peut aussi se noter :
`Élève (INE, Nom, Prénom, Âge, Sexe, #Classe)`
L'attribut sur lequel est posé une contrainte de clé primaire est souligné et l'attribut sur lequel est posé une contrainte de clé étrangère est précédé du caractère `#`.

On peut représenter le schéma ainsi :

Élève
INE
Nom
Prénom
Âge
Sexe
#Classe

Un schéma relationnel, ou schéma d'une base de données relationnelle, est un ensemble de schémas de relations liés par des attributs communs.

Conception d'une relation

Si à une valeur d'un attribut A correspond une et une seule valeur d'un attribut B, on dit que l'attribut B est en *dépendance fonctionnelle* de l'attribut A.

Par exemple, à un identifiant d'élève correspond un unique nom d'élève.

Dans l'élaboration des relations, il est important d'éviter la redondance des données et de faciliter leur mise à jour. Cela permet de garantir la cohérence des informations.

De manière générale, tous les attributs ne contiennent qu'une seule information et sont en dépendance fonctionnelle de la clé primaire. On n'utilisera pas, par exemple, un attribut `Nom Prénom` qui contient deux informations.

Si une clé primaire est constituée de plusieurs attributs, les autres attributs sont en dépendance fonctionnelle de l'intégralité de la clé primaire et pas seulement d'une partie de celle-ci.

De plus, tous les attributs sont en dépendance fonctionnelle uniquement de la clé primaire.

Organisation des données

Les données peuvent être représentées dans un tableau comme celui-ci :

INE	Nom	Prénom	Âge	Sexe	Classe
1	Turing				2
2	Hopper				4
3	Codd				1
...					
50	Bachman				
...					

Remarque : ce tableau ne doit pas être interprété comme celui obtenu dans un tableur.

Un tableur permet d'effectuer diverses actions sur ces données : des calculs, des schémas, des graphiques. C'est principalement un moyen de stocker ces données. On l'utilise d'ailleurs pour recueillir des informations provenant d'une base de données ou pour communiquer des informations à une base de données. Mais il n'y a aucune modélisation, pas de liens entre les données. Un texte peut être remplacé par un nombre, des données peuvent être modifiées ou supprimées, il n'y a pas d'accès multi-utilisateurs. Ces questions ne sont pas gérées par le tableur.

Chaque case de ce tableau contient une information unique. Par exemple, on n'inscrit pas deux prénoms dans une case de la colonne `Prénom`.

L'attribut `INE` est un identifiant numérique qui permet d'identifier chaque ligne de manière unique. En effet, on n'est pas certain de pouvoir identifier une ligne uniquement à l'aide du nom, car plusieurs élèves peuvent avoir le même nom, ni même en utilisant le nom et le prénom.

Cet attribut `INE` peut donc être défini comme clé primaire. On dit qu'on pose *une contrainte de clé primaire sur l'attribut `INE`*.

Pour la deuxième relation nommée `Classe`, on a le tableau :

Nom	Effectif	Salle
PSI1	45	B307
PC	38	C206
PSI2	42	B311
PT1	35	B205
...	...	...

Deux tableaux représentent les données qui avec un tableur pourraient être stockées dans un tableau unique.

Pour distinguer les attributs `Nom` des relations `Élève` et `Classe`, nous pouvons utiliser la notation pointée comme `Élève.Nom` et `Classe.Nom`.

L'attribut `Nom` de la relation `Classe` est une clé primaire. Nous souhaitons poser une contrainte de clé étrangère sur l'attribut `Classe` de la relation `Élève`.

Or, pour effectuer des recherches et pour procéder à d'éventuelles modifications, il est plus efficace d'utiliser un nombre entier plutôt qu'une chaîne de caractères. On ajoute donc une colonne

supplémentaire, un nouvel attribut nommé `Id`. Il s'agit d'une clé artificielle ou clé de substitution (*surrogate key* en anglais) qui permet d'identifier une ligne quelconque.

Id	Nom	Effectif	Salle
1	PSI1	45	B307
2	PC	38	C206
3	PSI2	42	B311
4	PT1	35	B205
...	...	...	...

Les valeurs de la clé étrangère `Classe` de la relation `Élève` sont alors des valeurs de la clé primaire `Id` de la relation `Classe`.

Avec cette représentation, les modifications sont beaucoup plus simples à gérer que si toutes les données étaient dans un tableau unique. Ainsi, si par exemple les élèves de la classe PSI1 changent de salle, il n'y a qu'une seule valeur à modifier dans le tableau `Classe`.

Remarque : une relation est bien autre chose qu'un simple tableau. Nous pouvons permuter les colonnes ou les lignes sans modifier la relation qui est un ensemble de n-uplets. Par exemple la relation $R = \{(a, x), (b, x), (c, y)\}$ peut aussi s'écrire $R = \{(x, b), (y, c), (x, a)\}$.

La notion de contrainte est capitale. Supposons que dans la table `Classe` un enregistrement, ou une ligne, contient la valeur 7 pour l'identifiant `Id` et que cette valeur 7 est celle d'un ou plusieurs enregistrements pour l'attribut `Classe` dans la table `Élève`.

Si par erreur, la suppression dans la table `Classe` de l'enregistrement avec l'identifiant 7 est demandée, cela va provoquer une erreur car cela conduirait à une violation de la contrainte de référence, ou contrainte de clé étrangère, de `Élève.Classe` vers `Classe.Id`.

En pratique, c'est généralement le comportement par défaut. Mais il peut être modifié en imposant par exemple une suppression en cascade. Dans ce cas, toutes les lignes concernées de la table `Élève` sont supprimées. On peut aussi imposer que les lignes concernées ne soient pas supprimées mais qu'une valeur par défaut remplace la valeur 7.

■ Bases de données relationnelles et SGBD

Dans une base de données, les informations doivent pouvoir être facilement consultables et mises à jour. Il existe plusieurs types de bases de données. Une base de données relationnelle utilise le modèle relationnel. Un SGBD est un système de gestion de base de données. L'informaticien américain Charles Bachman est le premier concepteur d'un SGBD moderne.

■ SGBD

Pour accéder à une base de données, on utilise donc un SGBD. Ce peut être un composant logiciel, comme SQLite qui est une bibliothèque avec une interface directement intégrable dans un programme. D'autres SGBD, comme MySQL et PostgreSQL, fonctionnent avec un serveur.

Les systèmes de gestion de base de données sont de deux types :

- des systèmes libres comme MySQL, PostgreSQL, SQLite ;
- des systèmes propriétaires comme Oracle Database de l'entreprise Oracle, SQL Server de Microsoft, Access qui fait partie de la suite Microsoft Office.

Le concept client-serveur

De manière générale, un utilisateur (une personne ou une machine, le *client*) émet des requêtes pour accéder aux données, les modifier, les supprimer, en créer d'autres. Ces requêtes sont envoyées depuis une machine distante du *serveur* qui est le lieu où les données gérées par un SGBD sont stockées physiquement. La machine de l'utilisateur doit accéder à un réseau auquel est connecté ce serveur. C'est une application client qui communique avec le SGBD et envoie les demandes de l'utilisateur.

Fonction d'un SGBD

- Le système doit assurer la *persistance des données*. Ceci consiste à garder en mémoire des versions antérieures lorsque des modifications sont effectuées. Avec une structure de données persistante, les modifications ne sont pas effectuées en place, la structure est immuable.
- Le système doit gérer des accès concurrents et la sécurité. Il gère les droits et privilèges d'un utilisateur ou d'un groupe d'utilisateurs sur un ensemble d'objets. Concernant la sauvegarde et la restauration des données, il existe différentes stratégies et ce sont des activités complexes.
- L'un des attendus principaux est un accès aux données efficace. Lorsque ces données se comptent par milliers ou par millions, il faut disposer d'algorithmes performants.

Remarque : un système de gestion de fichiers sur un ordinateur ou un smartphone n'a pas ces contraintes. L'accès est réservé à un utilisateur qui peut modifier, ajouter, supprimer des données dans un fichier. Mais il y a une idée importante, commune aux deux systèmes : l'utilisateur n'a pas besoin de connaître les processus logiques de fonctionnement ni l'aspect physique du stockage des fichiers sur la machine, ni la façon dont sont enregistrées les données dans les fichiers.

L'efficacité est conditionnée par la manière de stocker les différentes clés primaires. C'est un préalable à l'obtention depuis une table des valeurs d'un n-uplet qui est défini par une valeur de la clé primaire. L'utilisation d'une clé de substitution apporte une simplification pour la recherche. On perd de la place en ajoutant un attribut mais on gagne en rapidité d'accès.

Si on utilisait une structure linéaire pour stocker les clés, des algorithmes de recherche et de tri permettraient d'accéder à des clés particulières. On pourrait utiliser une recherche dichotomique dans le cas d'une séquence triée.

Mais il existe une structure permettant des algorithmes performants, c'est la structure d'arbre, en particulier la structure d'arbre binaire de recherche dans lequel une recherche est similaire à une recherche dichotomique. Les SGBD utilisent une structure encore plus adaptée, la structure de *B-arbre* (*Btree* en anglais) qui généralise la structure d'arbre binaire de recherche, ainsi que des tables de hachage (présentées au chapitre 3).

■ Entités-Associations

Pour concevoir une base de données relationnelle, il faut commencer par créer un modèle pour le schéma de la base de données et il peut y avoir plusieurs possibilités.

Un modèle conceptuel consiste à décrire les informations représentant les données de manière simple. Le modèle conceptuel est ensuite traduit en modèle relationnel. On représente un élément qui joue un rôle dans le système que l'on décrit par une *entité*. Cet élément peut être physique, concret, ou abstrait. Une entité est donc un objet possédant des propriétés appelées attributs servant à le décrire. Chaque attribut possède une valeur simple. Et l'ensemble des valeurs des attributs constituent un n-uplet.

L'ensemble des objets qui ont des caractéristiques communes, les attributs, est un *type d'entité* ou une *classe d'entité*. Un attribut particulier possède une valeur différente pour chaque entité et

permet donc de distinguer deux entités. On l'appelle *identifiant*. Chaque classe d'entité possède au moins un attribut identifiant.

On représente un type d'entité avec son nom et les noms des différents attributs dans un tableau comme ceux présentés dans les exemples qui suivent.

Par exemple, considérons un type d'entité `Élève` avec les attributs `INE`, `Nom`, `Prénom`, `Âge`, `Sexe`. L'attribut `INE` est l'identifiant que l'on peut souligner.

Élève
INE
Nom
Prénom
Âge
Sexe

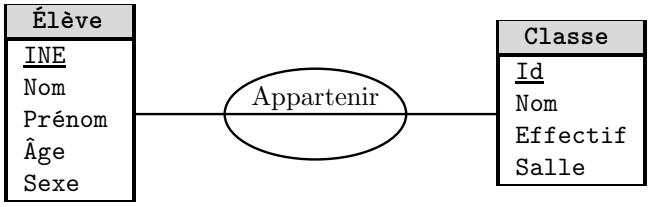
On peut considérer un deuxième type d'entité : le type `Classe` avec les attributs `Id`, `Nom`, `Effectif`, `Salle`. L'attribut `Id` est l'identifiant.

Classe
Id
Nom
Effectif
Salle

Des liens ou relations existent entre les entités. Par exemple nous avons une relation entre un élève et une classe. Une *classe de relation* entre deux types d'entités contient toutes les relations du même type entre les entités appartenant à ces deux types. Nous avons donc une classe de relation entre le type `Élève` et le type `Classe`. La relation ou le lien entre entités s'appelle une *association*.

Le *type d'association* est décrit par un titre comme `Appartenir` puisqu'un élève appartient à une classe ou `Contenir` puisqu'une classe contient un élève.

On parle donc de modèle *entité-association*.



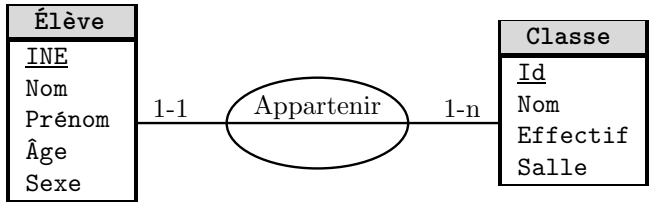
Le lien entre une entité et une association est caractérisé par la *cardinalité*. Celle-ci est un couple minimum-maximum qui indique le nombre minimum de fois et le nombre maximum de fois qu'une entité peut participer à l'association.

- Le couple est `1-1` si chaque entité participe exactement une fois à la relation.
- Le couple est `1-n` si chaque entité participe au moins une fois à la relation.
- Le couple est `0-n` si certaines entités ne participe pas à la relation.

Nous considérons l'exemple avec les types d'entité `Élève` et `Classe` et l'association ou le lien nommé `Appartenir`.

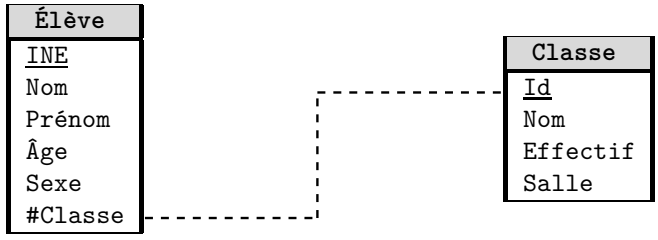
Il s'agit de déterminer les cardinalités qui caractérisent respectivement le lien entre `Élève` et `Appartenir` et le lien entre `Classe` et `Appartenir`.

- Le couple `1-1` entre **Élève** et **Appartenir** signifie que chaque élève appartient à exactement une classe.
- Le couple `1-n` entre **Appartenir** et **Classe** signifie que chaque classe contient au moins un élève.

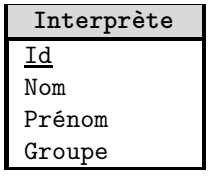


On peut présenter le modèle entité-association en parlant d'associations `1-*`, `1-1`, ou `*-*` entre deux entités. Par exemple, nous avons une association `1-*` entre **Élève** et **Classe**. Ceci signifie qu'un élève appartient à une classe et qu'une classe peut contenir plusieurs élèves.

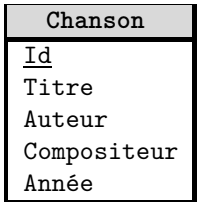
Dans le modèle relationnel, les deux types d'entités deviennent des tables **Élève** et **Classe** avec leurs attributs respectifs et les identifiants respectifs `INE` et `Id` appelés clés primaires. L'association `1-*` conduit à ajouter un attribut à la table **Élève** et à poser sur cet attribut une contrainte de clé étrangère en relation avec la clé primaire `Id` de la table **Classe**.



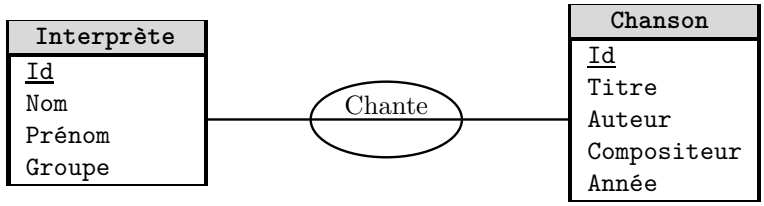
Considérons un autre exemple avec un type **Interprète** possédant les attributs **Id**, **Nom**, **Prénom**, **Groupe**. L'attribut **Id** est l'identifiant.



Un deuxième type est le type **Chanson** avec les attributs **Id**, **Titre**, **Auteur**, **Compositeur**, **Année**. L'attribut **Id** est l'identifiant.

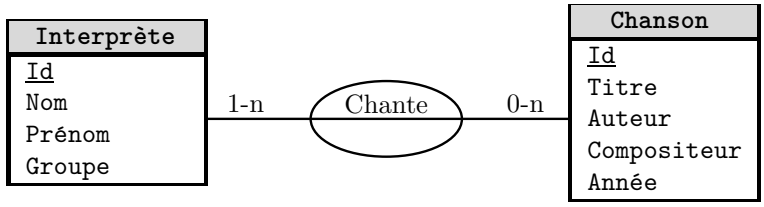


Nous avons une relation entre un interprète et une chanson donc une classe de relation entre le type **Interprète** et le type **Chanson**. Cette association est décrite par un titre comme **Chante** puisqu'un interprète chante une chanson ou bien par un titre comme **Est chantée par** puisqu'une chanson est chantée par un interprète.



Traduisons le nombre de fois qu'une entité peut participer à l'association.

- le couple `1-n` entre `Interprète` et `Chante` signifie que chaque interprète chante au moins une chanson.
- le couple `0-n` entre `Chante` et `Chanson` signifie que certaines chansons peuvent être chantées par aucun interprète et d'autres par un ou plusieurs interprètes.

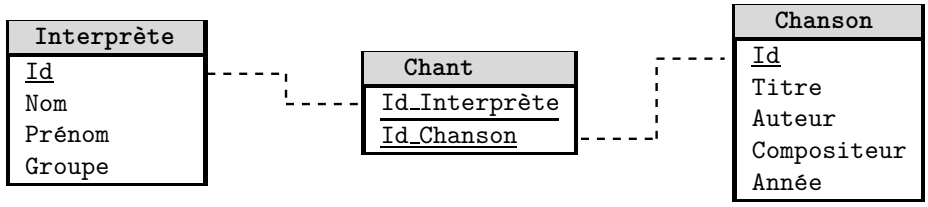


Nous avons une association `*-*` entre `Interprète` et `Chanson`. Ceci signifie qu'un interprète peut chanter plusieurs chansons et qu'une chanson peut être chantée par plusieurs interprètes.

Une association `*-*`, comme entre `Interprète` et `Chanson`, est décomposée en deux associations `1-*`, une entre `Chante` et `Interprète` et une entre `Chante` et `Chanson`.

La traduction de ce modèle conceptuel des données en un modèle relationnel nécessite donc, dans le cas de cet exemple, trois tables : une table `Interprète`, une table `Chanson` et une table `Chant`. Un type d'entité est traduit par une table avec pour clé primaire l'identifiant du type d'entité. La troisième table `Chant` a pour clé primaire le couple (`Id_interprète`, `Id_chanson`).

Le type d'association `Chante` est donc traduit par une table en prenant par exemple pour clé primaire le couple composé des clés primaires des deux entités. Les attributs `Id_interprète` et `Id_chanson` sont des clés étrangères en relation respectivement avec la clé primaire `Id` de `Interprète` et la clé primaire `Id` de `Chanson`.



Une association `1-1` signifierait que chaque interprète ne chante qu'une chanson et que chaque chanson n'est chantée que par un interprète.

Une association entre deux entités de type `1-1` est traduite pour chaque table par la présence d'une clé étrangère en lien avec la clé primaire de l'autre table.

Remarque 1 : on peut comparer une entité ou une relation à un type ou une classe d'objet en Python. Une instanciation de l'entité ou un enregistrement de la relation correspond alors à une variable ou un objet.

Remarque 2 : la modélisation est dépendante du traitement que l'on souhaite effectuer. On peut par exemple considérer un type d'entité `Ville` avec divers attributs (comme le nombre d'habitants, le département, etc) ou considérer `Ville` comme un attribut d'un type d'entité `Adresse`.

Remarque 3 : les associations des types `1-1` et `1-*` se traduisent dans les tables à l'aide de clés primaires et de clés étrangères. Celles-ci sont évoquées en termes de contraintes dans les bases de données. Elles sont utilisées dans la notion de jointure en langage SQL.

Vrai/Faux

	Vrai	Faux
1. Une relation est un ensemble de n-uplets.	☐	☐
2. Une clé primaire est toujours composée d'un seul attribut.	☐	☐
3. Une contrainte d'intégrité permet d'assurer la cohérence dans les données.	☐	☐
4. La persistance des données signifie que les données ne sont jamais effacées.	☐	☐
5. Edgar Franck Codd a conçu le premier SGBD moderne.	☐	☐
6. Les valeurs prises par un attribut sont toutes du même type.	☐	☐
7. Les valeurs prises par une clé étrangère sont strictement les mêmes que celles prises par la clé primaire avec laquelle elle est en relation.	☐	☐
8. Une association `1-*` entre deux entités se traduit par deux tables.	☐	☐
9. Une association `1-*` se traduit par deux tables qui contiennent chacune une clé étrangère.	☐	☐
10. Supprimer une valeur d'une clé primaire provoque toujours une erreur s'il existe une clé étrangère en relation avec cette clé primaire.	☐	☐