1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

26 INTRODUCTION AUX BASES DE DONNEES ET AUX SYSTEMES DE GESTION DE BASE DE DONNEES

26.1 BASE DE DONNEES

Selon Chris L. Date [Dat95] une base de données (B.D.) consiste en une collection de données **persistantes** utilisées par des systèmes d'application de certaines **organisations**.

Deux mots mots clés sont utilisés dans cette définition :

- Données persistantes : elles diffèrent d'autres données plus éphémères telles que les données d'entrée, les données de sortie et plus généralement de toute donnée de nature transitoire.
- Organisations : peut être un simple individu avec une petite base de données privée (un médecin, un notaire, …) ou une société complète avec une base de données partagée très importante (une banque, une université, un hôpital,…).

Les avantages d'utilisation d'une base de données par rapport aux systèmes traditionnels (fichiers liés à l'application) sont nombreux.
Dans le cas d'une utilisation par un simple individu, nous pouvons citer :

- la compacité : plus besoin de fichiers manuels volumineux, fiches cartonnées par exemple où le risque de perte d'information est important,
- la rapidité : recherche et mise à jour rapide d'information

Ces avantages sont encore plus grands quand il s'agit d'une base de données importante. Dans ce cas les données sont

- intégrées, i.e., la base de données est vue comme un regroupement de plusieurs fichiers de données où toute redondance a été totalement ou partiellement éliminée. Une conséquence importante de cet avantage est la diminution de l'incohérence. En effet, dans le contexte classique, les fichiers sont conçus pour satisfaire les besoins d'une application. Il s'ensuit qu'une même information peut être représentée dans plusieurs fichiers. Si elle intervient dans deux applications, elle peut même se présenter différemment (format, codage,…). Cela entraîne un risque de conflit : lors d'une mise à jour, par exemple, tous les duplicatas ne sont pas mis à jour en même temps par suite de leur non appartenance à l'application qui réalise cette mise à jour.
- partagées, i.e., les différents utilisateurs peuvent faire usage de ces données pour des buts divers. Ils peuvent également accéder aux mêmes données simultanément. Ce type de données est en faite une conséquence de l'intégration des données.

26.2 NIVEAUX DE REPRESENTATION DES DONNEES

Trois niveaux sont proposés par le groupe ANSI/SPARC pour l'architecture d'une base de données **figure1.**

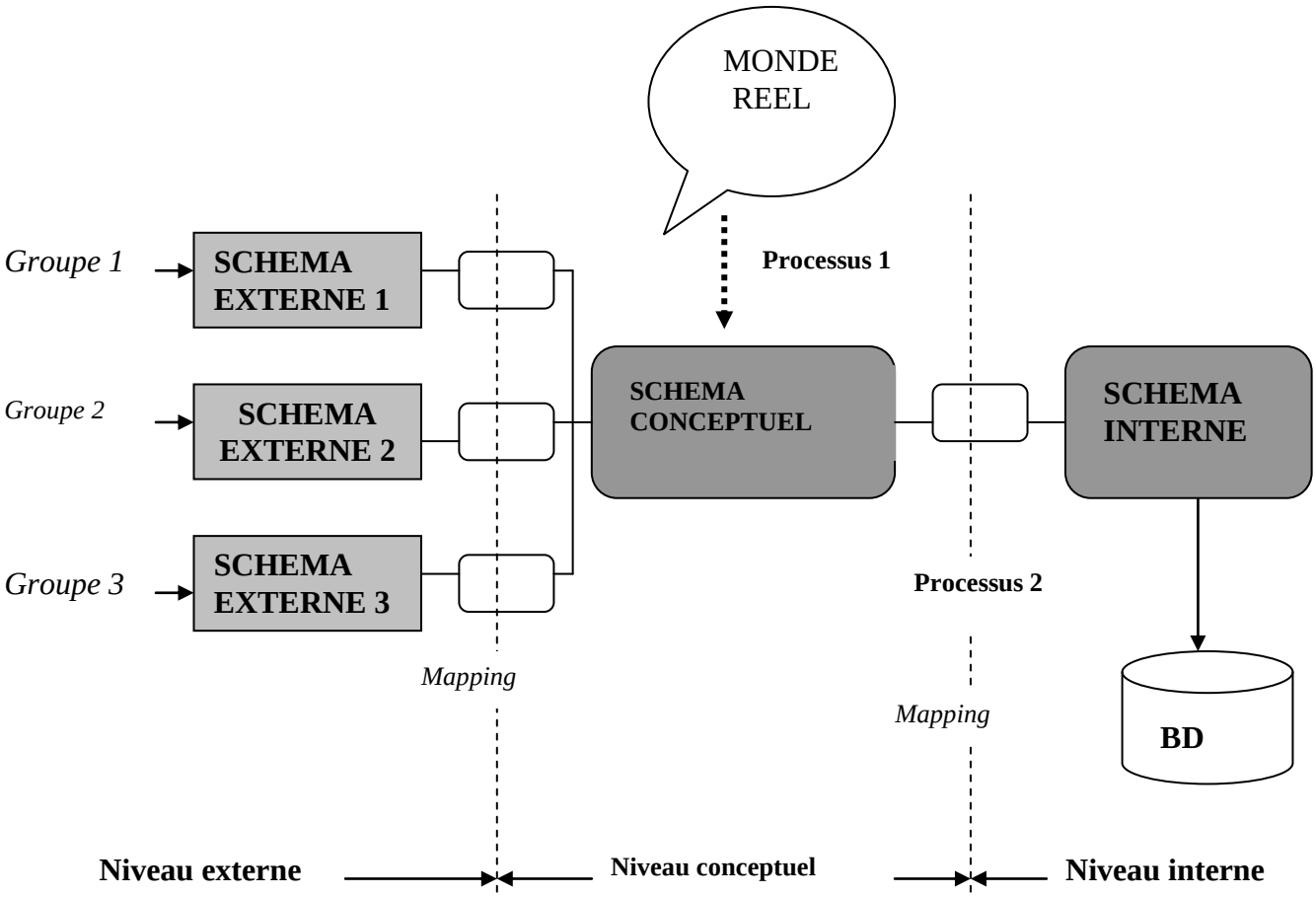


Fig 1. Niveaux de Bases de données

26.2.1 Le niveau conceptuel

Le schéma conceptuel est la « charpente » d'une BD. Il décrit en termes abstraits la réalité organisationnelle et ses règles de gestion.
Le processus de conception consiste à traduire les objets du monde réel en catégories d'objets suivant des **MODELES** bien définis.
Il existe plusieurs types de modèles que nous pouvons classer en trois catégories :

- **Les modèles de 1ère génération : décennie 60**
  - HIERARCHIQUE
  - RESEAU
- **Les modèles de 2ième génération : les décennies 70 et 80**
  - E/A : ENTITE /ASSOCIATION
  - RELATIONNEL
  - LES RESEAUX SEMANTIQUES
- **Les modèles de 3ième génération : la décennie 90**
  - LE MODELE OBJET

26.2.2 Le niveau externe

Un schéma externe appelé aussi **VUE** dans les systèmes relationnels est une perception des données par un programme d'application. Une vue est un **SOUS-SCHEMA** d'un schéma conceptuel.
Le schéma externe peut contenir des informations complémentaires (par exemple des informations de calcul). Les différents schémas externes permettent la validation du schéma conceptuel. D'ailleurs il existe une technique de construction du schéma conceptuel à partir de l'intégration de vues.

26.2.3 Le niveau interne

Il est le niveau relatif à la mémoire physique, i.e., il concerne la manière selon laquelle les données sont réellement stockées, il correspond au schéma interne.

En d'autres termes, il existera plusieurs vues externes distinctes, chacune d'elles correspond à une représentation plus ou moins abstraite d'une partie de la base de données et il n'existera qu'une seule vue conceptuelle qui donnera une représentation abstraite de la totalité de la base de données. De même qu'il n'existera qu'une seule vue interne représentant la totalité de la base telle qu'elle est enregistrée en mémoire.
Par ailleurs, le processus de développement d'une base de données comprend deux grandes phases :

- Une phase de **conception** dont le résultat est le **SCHEMA CONCEPTUEL**. Au cours de cette phase, la validation s'effectue à travers les **SCHEMAS EXTERNES**
- Une phase de réalisation dont le résultat est le **SCHEMA INTERNE** et la **BASE DE DONNEES**

26.3 SYSTEME DE GESTION DE BASE DE DONNEES.

Un système de gestion de base de données (SGBD) est un logiciel qui prend en charge tous les accès à la base de données; généralement il répond aux objectifs suivants que nous reprendrons plus en détail dans les chapitres suivants.

- **Indépendance des données.**

Il existe deux types d'indépendance : physique et logique

1. Indépendance physique. La notion opposée, la dépendance des données, peut mieux expliquer cette notion. Les applications implantées sur les anciens systèmes ont tendance à être plutôt à données dépendantes : la manière de stocker les données en mémoire secondaire et les techniques d'accès à ces données sont complètement dictées par les besoins de l'application considérée. Par exemple, si un ensemble de données est organisé sous une forme qui utilise une table d'index pour accéder aux données, le programmeur qui souhaite traiter des données relatives à cet ensemble construira son programme en fonction de la connaissance qu'il a de l'organisation physique. Dans ce cas, nous dirons que le SGBD n'a pas d'indépendance physique parce qu'il n'est pas possible de changer l'organisation physique sans entraîner une modification des programmes d'application. L'indépendance physique permet donc à l'administrateur de la base de données de modifier, pour des raisons d'efficacité, l'organisation physique des données sans modification des programmes. Par exemple, il doit pouvoir ajouter une table d'index,

accélérant ainsi l'accès aux données, sans toucher les programmes d'application déjà existants.
2. Indépendance logique. Elle permet de modifier le schéma conceptuel, par exemple en ajoutant de nouvelles classes d'objets ou bien de nouvelles associations, sans modifier les programmes d'applications.

L'indépendance des données est un des objectifs majeurs des SGBD. Elle peut être définie comme l'immunité des applications face à tout changement survenant dans la structure de mémorisation ou dans la technique d'accès.

- **Définition des données**

Cet objectif permet de définir les données (schémas externes, schéma conceptuel, schéma interne et tous les liens correspondants) sous une forme non compilée et de les convertir dans la forme objet appropriée. Le SGBD doit donc être muni d'un langage de définition de données (LDD).

- **Manipulation de données**

Cet objectif permet de traiter les requêtes de l'utilisateur pour interroger, modifier, supprimer ou insérer de nouvelles données. Dans ce cas, le SGBD est muni d'un langage de manipulation de données (LMD).

Les requêtes peuvent être prévues ou non prévues
1. Les requêtes prévues sont généralement intégrées dans un programme d'application écrit en langage hôte[1]. Dans ce cas, l'administrateur de la base de données fait en sorte que la conception physique de la base de données garantisse de bonnes performances pour de telles requêtes.
2. Les requêtes non prévues sont généralement émises en interactif, car le besoin ne s'est pas fait sentir d'avance. Dans ce cas, la conception physique de la base de données peut ne pas avoir été conçue de manière à répondre efficacement à de telles requêtes. Ce problème représente un défi pour les concepteurs actuels de SGBD.

- **Efficacité des accès aux données**

Les accès aux données seront plus efficaces que dans les SGF grâce notamment :
1. Au développement d'index sophistiqués
2. A l'existence de plusieurs chemins d'accès à une donnée
3. A l'existence de techniques d'optimisation de requêtes qui sélectionnent le chemin optimal à une donnée.

- **Cohérence**

Les données de la base obéissent à des règles appelées **contraintes d'intégrité (CI).**
Une CI est une assertion que doit vérifier le SGBD à chaque fois que la donnée sur laquelle elle est définie est sollicitée (déchargement de l'utilisateur lors des opérations de création, modification, suppression).

[1] Un langage hôte est un langage traditionnel (COBOL, PL1, C…) auquel on intègre des instructions du LMD.

Une base de données cohérente est une base de données dont les contrainte d'intégrité sont toujours vérifiées lors d'opérations sur la base de donnée.

- **Partageabilité**

Le SGBD doit permettre à plusieurs applications de partager les données. Pour cela, il doit gérer les conflits d'accès (les détecter et les solutionner).
Ceci est possible grâce à la notion de **transaction** et de définition **d'algorithmes de gestion de la concurrence.**

Une transaction est un ensemble indécomposable d'opérations sur la BD dont l'exécution maintient la BD dans un état cohérent.

- **Sécurité et confidentialité**

Les données doivent être protégées contre les pannes et contre les accès mal intentionnés.

- **Protection contre les pannes**

On distingue deux types de pannes :

- Pannes simples caractérisées par la perte du contenu de la mémoire centrale.
- Pannes catastrophiques caractérisée par la perte du contenu des mémoires secondaires.

Dans les deux cas, il est possible de récupérer un état cohérent des données grâce à l'existence et à la gestion de journaux de transactions, qui gardent la trace d'exécutions antérieures.

- **Protection contre les accès mal intentionnés**

La définition de droits d'accès, de mots de passe etc. gérés par le SGBD permet la confidentialité des données de la base et leur inviolabilité.

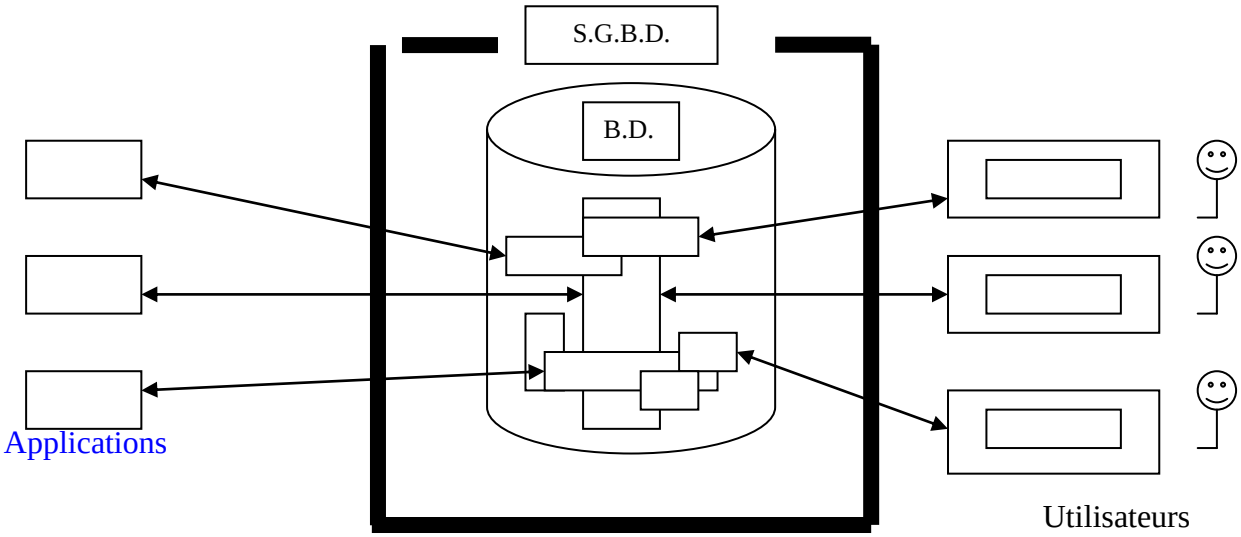


Fig. 2. BD et SGBD

26.4 DIFFERENTS MODELES DE DONNEES

Différents modèles de données existent liés aux différentes générations de bases de données. La première décennie (années 60) caractérise la première génération et est basée sur les modèles hiérarchiques et réseaux. La deuxième décennie (années 70) a vu l'émergence de modèles plus évolués, à savoir les modèles Entité/Association, Relationnel et Réseaux Sémantiques qui ont caractérisé la deuxième génération. Les années 90 ont vu apparaître des modèles permettant la modélisation de structures complexes de données, ce sont les modèles objets, qui caractérisent la troisième génération.

Pour comprendre l'apport du modèle relationnel, nous donnons dans cette section une description succincte des deux premiers modèles. Le modèle relationnel fera l'objet de la section suivante. Une présentation plus détaillée de ce modèle fera l'objet des chapitres suivants.

26.4.1 Modèle hiérarchique

Ce modèle est né après une tentative de généralisation des notions développées dans COBOL. Le système le plus représentatif est le SGBD IMS développé par IBM. Ce qui caractérise ce système est la très forte dépendance entre la description de la structure de données et la manière dont ces dernières sont enregistrées sur le support à accès direct.

Supposons que l'on veuille décrire, dans le cadre de l'application gestion des approvisionnements, une base de données décrivant l'association qui lie les fournisseurs aux pièces qu'ils commercialisent et inversement. Cet exemple de base de données emprunté et adapté de [DAT 95] sera notre principal exemple d'illustration dans les différents chapitres.
Sachant qu'un fournisseur est décrit par son numéro NF, son nom NOM, son code CODE et la ville où il est localisé VILLE. Qu'une pièce est décrite par son numéro NP, son nom NOM, son poids POIDS, son matériau MATERIAU et la ville où elle est stockée VILLE. On sait aussi qu'un fournisseur f fournit une pièce p en une certaine quantité q. Un fournisseur peut fournir plusieurs pièces et une pièce peut être fournie par plusieurs fournisseurs (association n-m). La contrainte de la hiérarchie (1-n) ne permet pas cette association. Selon les objectifs de l'application, le concepteur doit choisir entre le schéma (a) ou le schéma (b) de la figure 3

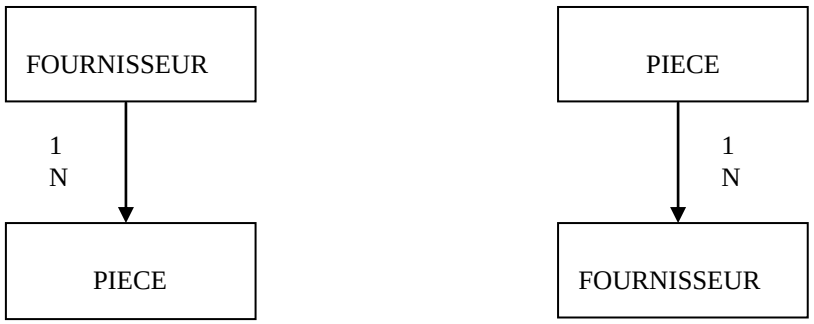


Fig. 3. Lien FOURNISSEUR - PIECE de la base de données approvisionnement dans un modèle hiérarchique

Mise en œuvre avec le modèle hiérarchique, l'association décrite dans Fig.3. pourrait se représenter par Fig. 4. Ci-dessous :

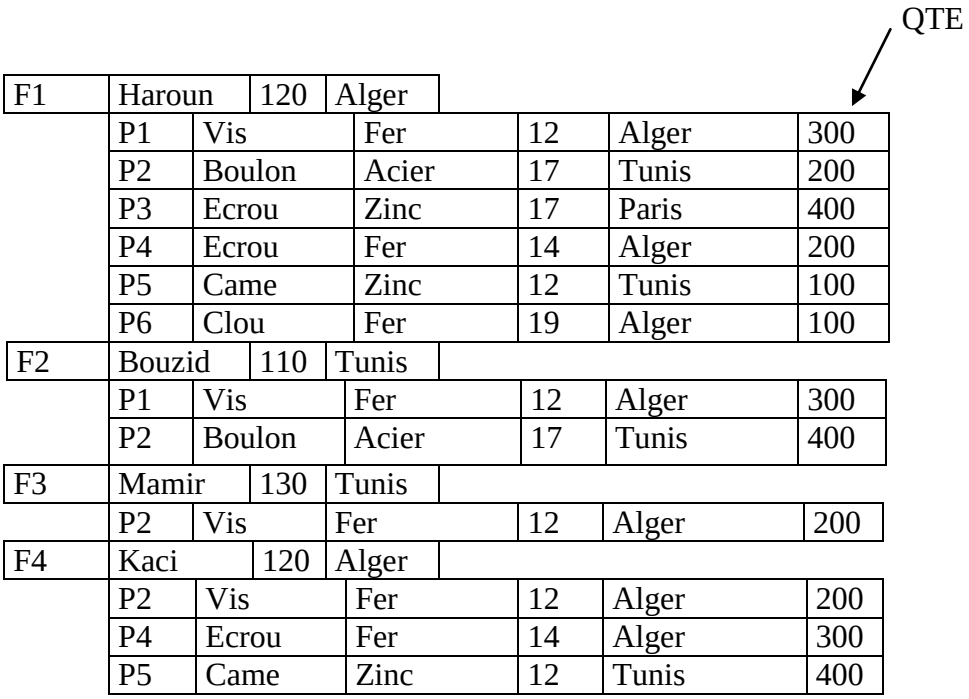


Fig. 4. Structure Hiérarchique

La figure Fig.4. fait apparaître quatre occurrences hiérarchiques, une pour chaque fournisseur. A chaque segment fournisseur (pour garder la terminologie IMS) est associé un ou plusieurs segments pièce. L'unité d'accès est le segment, et l'on ne peut accéder à un segment sans accéder à son supérieur hiérarchique. Ainsi si l'on veut établir la liste des pièces, il faudra analyser pour chaque fournisseur la liste des pièces fournies : on ne pourra donc pas faire abstraction de la présence des fournisseurs. Cette contrainte imposée par la hiérarchie pour accéder aux informations est très forte et très contraignante. Il s'ensuit qu'une mauvaise conception du schéma de la base de données peut être à l'origine de performances médiocres.
Cette dépendance entre la structure et la formulation des chemins d'accès apparaît nettement à travers un exemple simple. Considérons les deux requêtes suivantes avec leur formulation en DL/1 (nom du langage de manipulation de données IMS).

Q1 Trouver les pièces fournies par F2?

```
        GET UNIQUE fournisseur WITH
        NF = 'F2'
NEXT:  GET NEXT pièce FOR THIS
         fournisseur   pièce FOUND ? IF NOT

        Exit.
        PRINT NP.
        GO TO NEXT.
Exit.
```

Q2 Trouver les fournisseurs qui fournissent P2?

```
        GET TO START OF DATA
NEXT 1 : GET NEXT fournisseur
        Fournisseur FOUND? IF NOT, EXIT.
NEXT 2 : GET UNIQUE pièce FOR THIS fournisseur
              WITH NP = 'P2'
              Pièce FOUND ? IF NOT GO TO
              NEXT1
        PRINT NP.
        GET NEXT 2.
  Exit.
```

GET UNIQUE permet de rechercher le premier fournisseur répondant au critère de recherche NF = 'F2'. GET NEXT permet de passer à la pièce suivante.

On constate que si Q1 et Q2 sont symétriques quand au rôle joué par fournisseur et pièce, leur formulation ne l'est pas à cause de la hiérarchie qui impose une séquence d'analyse des données enregistrées dans la base. On constate que l'on n'a pas une formulation naturelle des questions qui permettent l'extraction de données de la base ce qui demande de la part des utilisateurs du SGBD une haute compétence technique. Par conséquent, cela rend difficile l'écriture, la mise au point et la maintenance des programmes d'application.
En ce qui concerne les opérations classiques de mise à jour (insertion, modification, suppression), la hiérarchie impose souvent l'utilisation d'artifices :

Insertion : si l'on veut enregistrer les caractéristiques d'une nouvelle pièce alors que celle-ci n'est fournie par aucun fournisseur, on doit introduire un fournisseur fictif.

Suppression : si l'on supprime le seul fournisseur d'une pièce particulière, l'information concernant cette pièce disparaît aussi. Ainsi la suppression de F1 entraîne la disparition des informations sur P6. La notion de hiérarchie implique que la suppression d'un segment entraîne celle de tous les segments subordonnés.

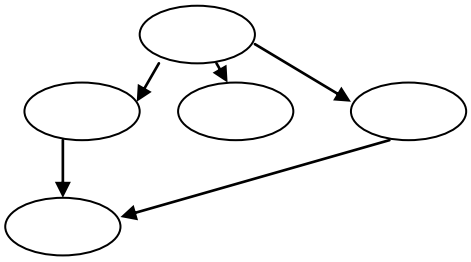
Modification : si l'on désire modifier le matériau de P2, il est nécessaire d'examiner toutes les listes de pièces fournies par les fournisseurs. On a donc un examen complet de la base.

Ce sont ces inconvénients, présentés de façon caricaturée, qui ont été à l'origine du développement des bases de données relationnelles où le premier objectif a été de supprimer ces contraintes imposées par la hiérarchie.

26.4.2 Modèle réseau

Un réseau est un ensemble de nœuds et d'arcs. Pour une représentation de données, les nœuds représentent les objets et les arcs, les associations entre ces objets.

Exemple :

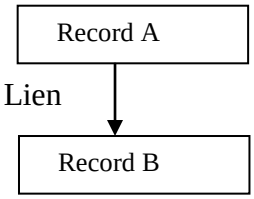


- Description des objets :

Les concepts définissant les objets sont les suivants : l'atome, l'agrégat, l'article ou enregistrement logique.

- Un atome (data item) est la plus petite unité de données possédant un nom. Un atome est représenté par une valeur dans la base
- Un agrégat (data aggregate) est une collection d'atomes rangés consécutivement dans la base et portant un nom.
- Un article ou enregistrement logique (record) est une collection d'agrégats et d'atomes rangés côte à côte dans la base de données et constituant l'unité d'échanges entre la base de données et les applications.

- Description des associations



Nous distinguons trois types de liens dans le modèle réseau :
1-1 : à un objet de A correspond un et un seul objet de B et vice versa
1-N : à un objet de A peut correspondre un à plusieurs objets de B mais inversement, à un objet de B ne peut correspondre qu'un seul objet de A (lien hiérarchique).
M-N : à un objet de A peut correspondre un à plusieurs objets de B et inversement.

Le modèle réseau CODASYL

En 1969 a eu lieu la conférence appelée CODASYL (Conference On Data System Language) dont un groupe, le DBTG (Data Base Task Group) a normalisé et présenté des concepts communs pour la définition de schémas réseaux. Tous les types d'objets représentés par des nœuds sont identifiés par des clés. Une clé est un atome ou un agrégat qui permet d'identifier de manière unique chaque record. De plus, le groupe a introduit le concept de sous-schéma, fondamental pour traiter les problèmes de sécurité, d'intégrité, de confidentialité et de partage des données.

Les types de liens supportés par la norme CODASYL sont les liens 1-1 et 1-N.
Le lien CODASYL est appelé SET. Un SET est un lien entre un enregistrement propriétaire appelé OWNER et un ou plusieurs enregistrements membres appelé MEMBER.
Un enregistrement ne peut être à la fois propriétaire et membre d'un même SET (pas de boucle). Cependant un enregistrement peut être propriétaire de plusieurs SET différents et ou membre de plusieurs SET différents.

Les liens M-N des modèles réseaux Fig.5 nécessitent des transformations que nous verrons plus loin.
Dans certains cas, le lien possède des propriétés qui relient les objets en relation. On les appelle des données d'intersection.

Exemple : Le lien entre fournisseur et pièce est un lien M-N dont une des propriétés d'intersection peut être la quantité.

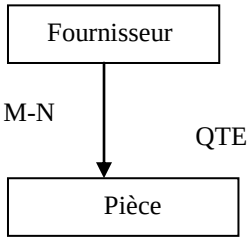


Fig. 5. Lien FOURNISSEUR - PIECE de la base de données approvisionnement dans un modèle réseau

- Transformation du lien M-N

Le lien M-N ne peut être représenté directement dans la norme CODASYL, étant donné que celle-ci ne prend en charge que les liens de type hiérarchique (owner-member).
Un artifice est utilisé qui consiste à créer un enregistrement intermédiaire, avec deux liens 1-N Fig. 6.

Exemple : Transformation du réseau fournisseur-piece

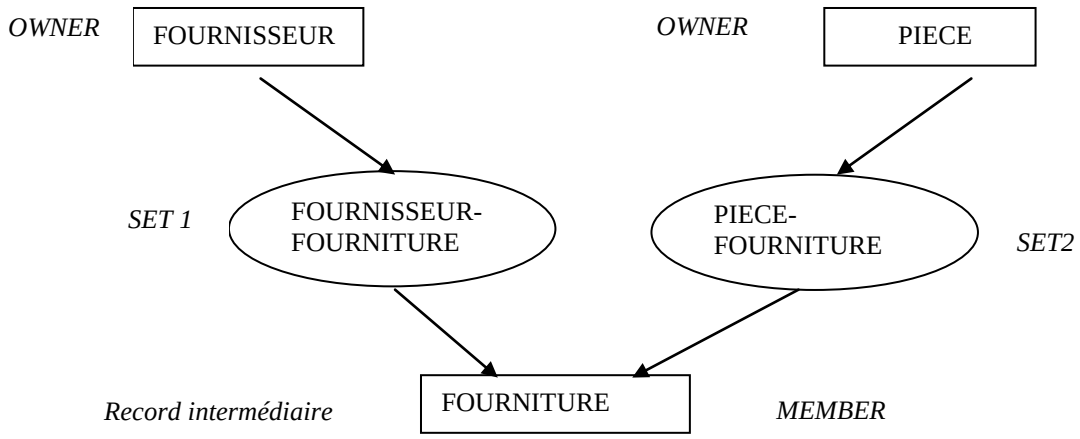
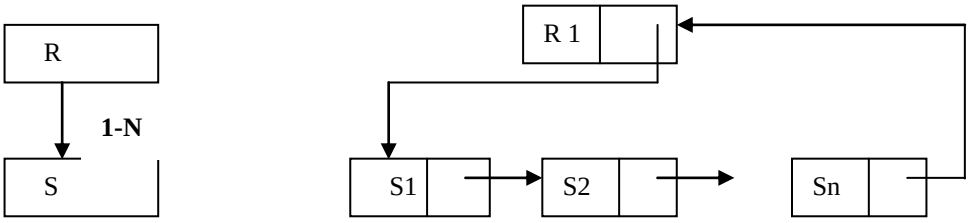


Fig. 6. Une représentation réseau de la base de données approvisionnement

Le record intermédiaire est crée en utilisant les données d'intersection si elles existent, et ayant pour clé la concaténation des clés des records composant le lien.
Pour décrire l'association M-N, FOURNISSEUR-PIECE, on crée le record intermédiaire FOURNITURE puis on associe FOURNISSEUR à FOURNITURE puis FOURNITURE à PIECE. On constitue ainsi des "sets".

Le lien CODASYL est implémenté grâce à la liste circulaire ou anneau appellé schéma d'occurrences.



Pour présenter d'une manière claire cette implémentation, nous allons restreindre la liaison fourniture qui nous sert d'exemple comme suit :

NF	NP	QTE
F1	P1	300
F1	P2	200
F1	P3	400
F2	P1	300
F2	P3	400

Fig. 5. Une extension restreinte de la liaison FOURNITURE

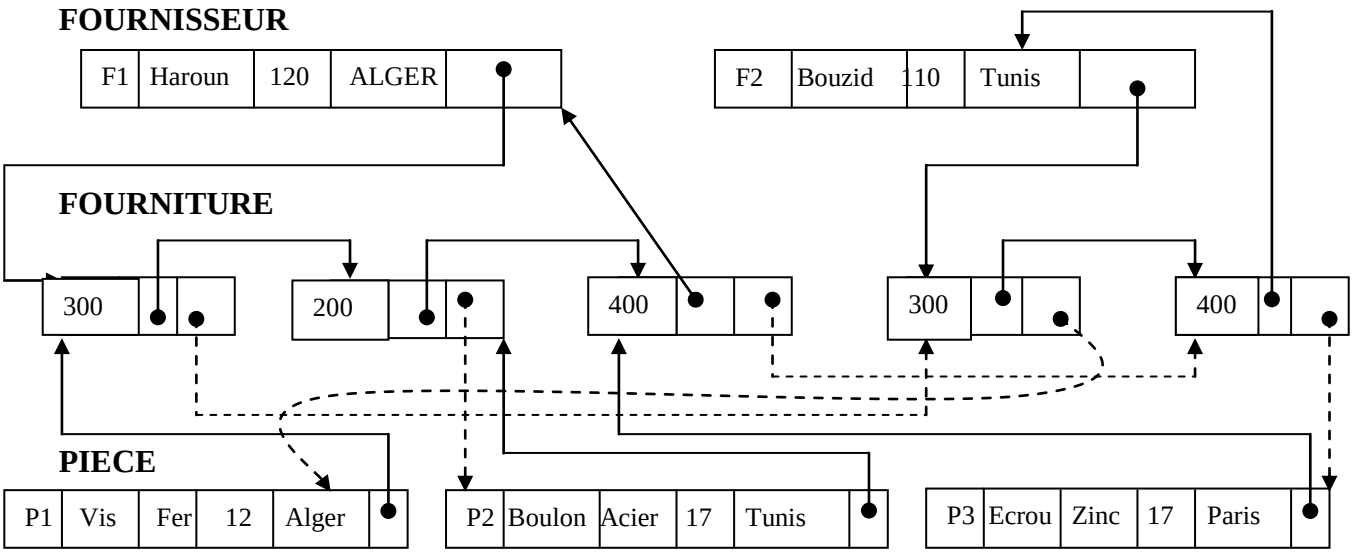


Fig. 7. Exemple d'implémentation : le schéma d'occurrences de la base de données exemple

Le langage de manipulation du modèle réseau doit permettre le parcours des anneaux qui relient les records d'un set. Il est réalisé à l'aide de l'instruction FIND qui possède plusieurs formats adaptés au type d'accès que l'on veut réaliser. L'instruction FIND permet de désigner un record et a pour rôle de positionner un pointeur (point courant) qui contient l'adresse du record sélectionné. La lecture de ce record est ensuite réalisée par l'instruction GET.

Nous allons, à titre de comparaison, reconsidérer les requêtes Q1 et Q2 précédentes.

Q1 Trouver les pièces fournies par F2	Q2 Trouver les fournisseurs qui fournissent P2
FIND Fournisseur WITH NF = 'F2'	FIND pièce WITH NP = 'P2'
NEXT : FIND NEXT LINK FOR THIS Fournisseur.	NEXT : FIND NEXT LINK FOR THIS pièce.
LINK FOUND? IF NOT, EXIT.	LINK FOUND? IF NOT, EXIT.
FIND Pièce FOR THIS LINK	FIND Fournisseur FOR THIS LINK
GET pièce.	GET fournisseur.
PRINT NP.	PRINT NF.
GO TO NEXT.	GO TO NEXT.
EXIT.	EXIT.

Nous constatons dans ce cas que des questions symétriques mettent en œuvre des algorithmes symétriques, ce qui est un avantage par rapport à l'approche hiérarchique. Mais cependant, comme dans le modèle réseau, on rencontre des problèmes de stratégie d'accès, notamment si l'on veut trouver la quantité des pièces P2 fournie par F2. Faut-il dans ce cas rechercher le fournisseur F2 et retrouver P2 ou inversement. On constatera généralement que selon le cas, les temps de réponse seront différents.

La réalisation des opérations classiques (insertion, suppression, et modification) est simplifiée par rapport au modèle hiérarchique :

Insertion : Il est trivial d'ajouter une nouvelle pièce. Initialement il n'y aura pas de liens.
Suppression : Il est possible de supprimer F1 sans P6 et inversement.
Modification : ne pose pas de problème particulier.

Le majeur inconvénient de la structure réseau réside dans la nécessité de parcourir des chaînages et de gérer des points courants. La gestion des points courants est très délicate à cause des interactions des instructions du LMD sur ces points courants. C'est cet aspect qui met en évidence l'avantage du modèle relationnel pour lesquels la notion de points courants n'existe pas au niveau du LMD qu'il met en œuvre.

26.5 LE MODELE ENTITE/ASSOCIATION DE CHEN 1976

Il est communément admis que la description de l'aspect statique de la réalité organisationnelle passe par la description de ses entités, de leurs propriétés, des liens entre les entités et des contraintes auxquelles elles sont soumises. Chen est parti de cet aspect descriptif pour proposer un modèle proche de la réalité. Il est basé sur les concepts ENTITE TYPE/ ASSOCIATION TYPE et sur deux formes d'abstraction : la classification et l'instanciation.

Une **association** est une combinaison d'entités dans laquelle chacune d'elle joue un rôle spécifique. La relation « père-fils » est une association entre deux personnes qui sont des **entités**. Les entités et les associations sont caractérisées par des **propriétés** : le nom de la personne, le total de la facture, la raison sociale de la compagnie ou la durée de la relation père-fils entre MOHAMED et ALI.
Les entités, associations et propriétés sont classées et définies par des types : l'entité type «Employé », l'association type « père-fils » ou la propriété type « Nom ». Un type définit en intension ou extension une population d'objets de la même nature. On parle aussi de classe d'entité ou classe d'association.
Une entité ou une association sont caractérisées par le doublet **« attribut valeur** »

Exemples

Employé :{e1} : (Nom, 'Ali'), (Date naissance, '251268'), (Salaire, '400000')
e1 est une instance ou occurrence du type ou de la classe Employé
Où Nom appartient au domaine Char (30)
Date naissance appartient au domaine Int(6)

Affectation(Employé, Projet) : (Datedeb, '010196'), (Datefin, '010199')

Le **schéma de l'entité type** définit l'entité type en **intention**.

Exemple :

Employé(Nom, Datenaissance, Salaire).

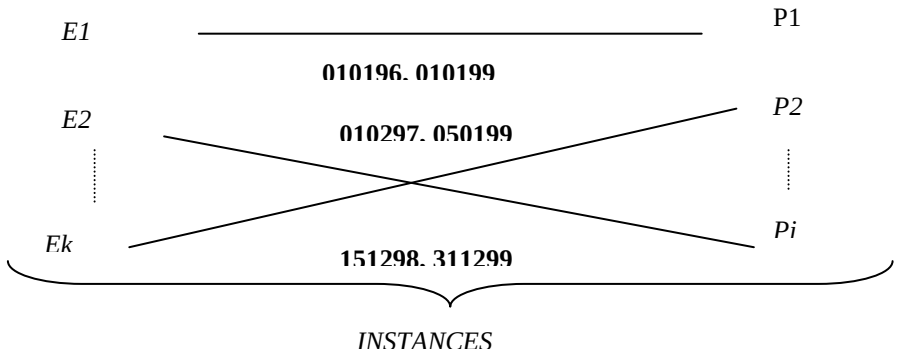
Un exemple d'extension de la classe (instances de la classe) est :
{Attaf, 251268, 40000}
{Benmohamed, 121065, 200000}
{Benmiloud, 050763, 100000}

Le **schéma d'association type** définit l'association type en intention.

Exemple

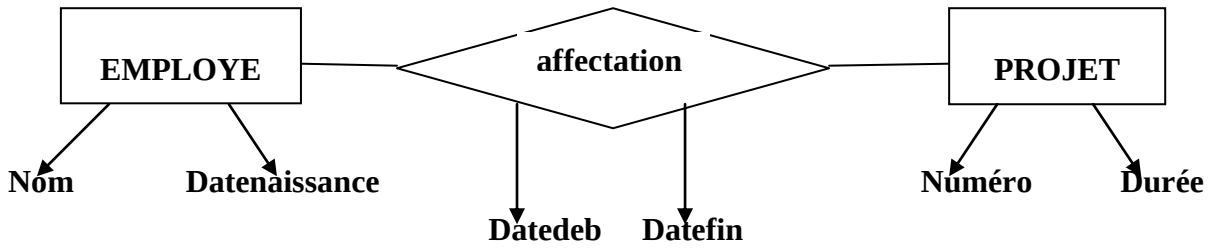
Affectation (Employé, Projet : Datedeb, Datefin)

Un exemple d'extension de la classe association est :

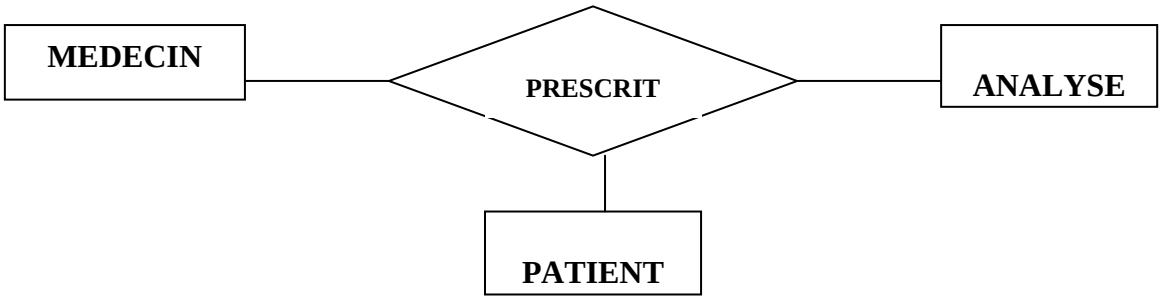


Graphiquement une entité type est représentée par un rectangle alors qu'une association type est représentée par un losange

Exemple 1 : association binaire



Exemple 2 : association N-aires

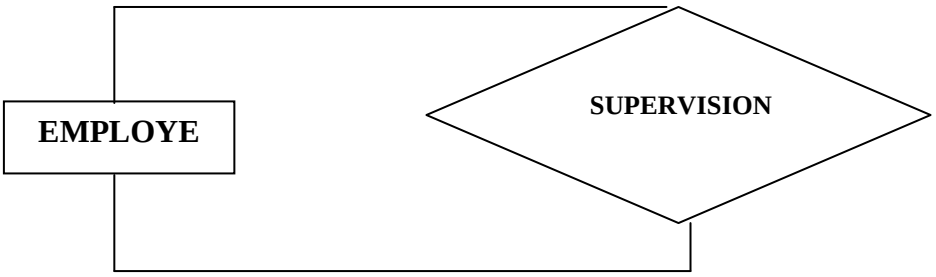


L'association prescrit relie un médecin à un patient et aux analyses qu'il doit faire.

Exemple3 : association récursives

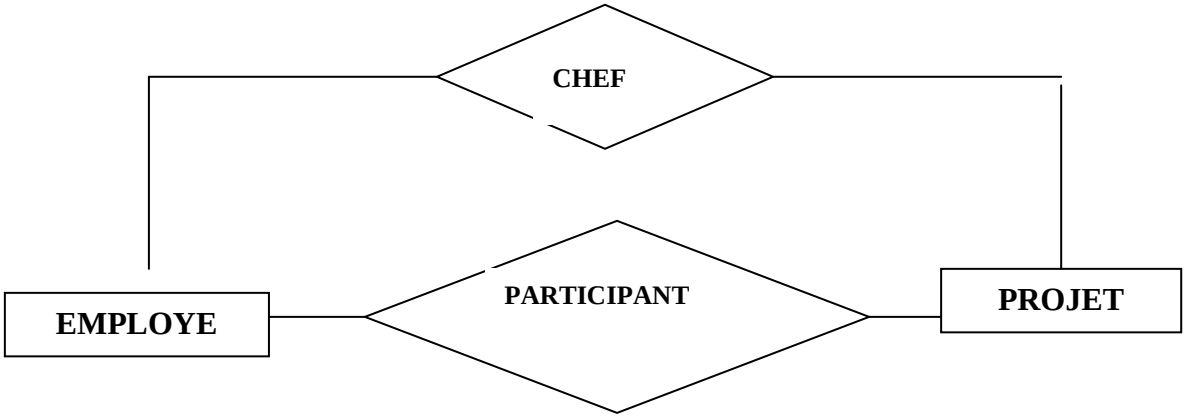
Une entité type peut jouer plusieurs rôles dans une association type

Rôle 1 : est supervisé



Rôle 2 : Superviseur

Exemple4 : Une entité peut jouer plusieurs rôles dans ses relations avec une autre entité type



- Une clé est un attribut ou groupe d'attribut dont les valeurs permettent d'identifier de manière unique les entités d'un même type.

La spécification de la clé d'une entité type implique une contrainte d'unicité sur l'extension de l'entité type.
Exemple :
FOURNISSEUR : NF
PIECE : NP

- La clé d'une association type est la combinaison des clés des entités types participantes

Exemple
AFFECTATION : NUEMP, NUMPROJ
FOURNITURE NF, NP
PRESCRIT : NUMEDECIN, NUMPATIENT, NUANALYSE

- La cardinalité est le nombre d'associations auxquelles une entité peut participer. Il existe trois cardinalité prédéfinies

- 1-1 : à une instance de l'objet A est associée une et une seule instance de l'objet B.
- 1-n : à une instance de l'objet A est associée une ou plusieurs instances de l'objet B.
- n-m : à une instance de l'objet A est associée une ou plusieurs instances de l'objet B et inversement.

Les raisons qui nous ont conduit à introduire ce modèle est qu'il fournit une aide utile à la conception d'une base de données. Nous allons voir dans le chapitre suivant que le modèle relationnel propose aussi une aide utile à la conception d'une base de données relationnelle et nous montrons dans le chapitre suivant, à l'aide de certaines règles de passage, qu'il est facile de passer d'une conception d'une base de données entités/associations à une base de données relationnelle et vice versa. Nous présentons brièvement dans la section suivante le modèle relationnel

26.6 MODELES RELATIONNELS

La plupart des recherches de ces 40 dernières années se sont basées sur l'approche relationnelle. Elle représente aujourd'hui la tendance principale du marché. La raison est que le modèle relationnel est fondé sur des bases mathématiques.
Brièvement, un système relationnel est un système dans lequel :

1. l'utilisateur perçoit les données comme des tables,
2. les opérateurs génèrent de nouvelles tables résultats à partir des tables existantes. Par exemples, des opérateurs à extraire certaines lignes ou certaines colonnes, qui sont elles mêmes des tables.

De tels systèmes sont appelés "relationnels" en raison du fait que le terme relation est le terme mathématique le plus approprié pour une table. Une illustration de base de données sous forme de tables est donnée par la figure 3 ci-dessous :

FOURNISSEUR

NF	NOM	CODE	VILLE
F 1	Haroun	120	Alger
F 2	Bouzid	110	Tunis
F 3	Mamir	130	Tunis
F 4	Kaci	120	Alger
F 5	Kaddour	130	Oran

PIECE

N P	NOM	MATERIAU	POIDS	VILLE
P 1	Vis	Fer	12	Alger
P 2	Boulon	Acier	17	Tunis
P 3	Ecrou	Zinc	17	Paris
P 4	Ecrou	Fer	14	Alger
P 5	Came	Zinc	12	Tunis
P 6	Clou	Fer	19	Alger

FOURNITURE

N F	N P	QTE
F 1	P 1	300
F 1	P 2	200
F 1	P 3	400
F 1	P 4	200
F 1	P 5	100
F 1	P 6	100
F 2	P 1	300
F 2	P 2	400
F 3	P 2	200
F 4	P 2	200
F 4	P 4	300
F 4	P 5	400

Fig. 8. La base de données Approvisionnement, une extension relationnelle

La table FOURNISSEUR décrit des fournisseurs où :

- NF est le numéro du fournisseur
- NOM est le nom du fournisseur
- CODE le code du fournisseur
- VILLE la ville où il se situe.

La table PIECE décrit les pièces où

- NP correspond au numéro de pièce
- NOM est le nom de la pièce
- MATERIAU, le matériau constituant la pièce
- POIDS le poids de la pièce
- VILLE la ville où est stockée la pièce.

La table FOURNITURE signifie qu'un fournisseur *f* fournit une pièce *p* en une certaine quantité q.

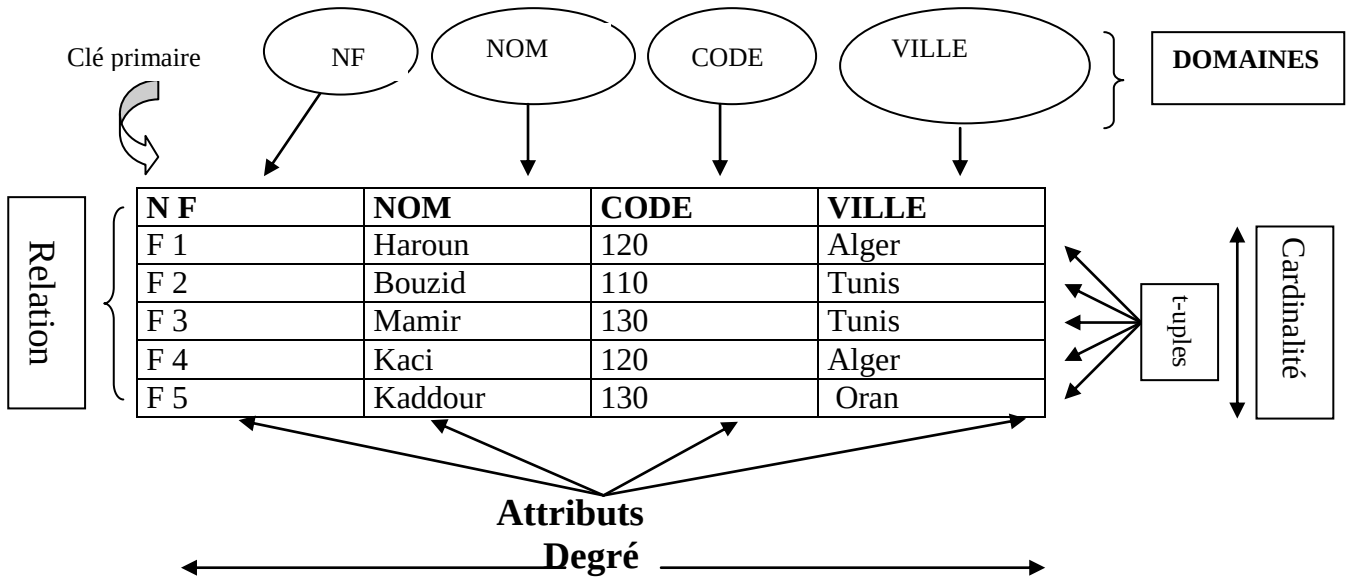


Fig. 4 : Terminologie d'une structure relationnelle

La plus petite unité sémantique de données (Valeur Atomique) telle que la valeur du numéro d'un fournisseur, du poids d'une pièce ou d'une quantité fournie par un fournisseur est appelée **scalaire**.

Un **domaine** est défini comme un ensemble donné de valeurs scalaires, toutes de même type. Par exemple, le domaine des numéros des fournisseurs, le domaine des villes …

Chaque **attribut** doit être défini sur exactement un seul domaine : les valeurs des attributs doivent être prises dans ce domaine. Par exemple, l'attribut NF dans la relation Fournisseur et l'attribut NF dans la relation Fourniture, vont tous deux être définis sur le domaine des numéros de fournisseur. Les valeurs de ces attributs sont des sous-ensembles des valeurs du domaine sur lequel ces attributs sont définis.

En fait, un domaine n'est rien d'autre qu'un type de données au sens langage de programmation.

Une **relations** R, sur un ensemble de domaines D1, D2,…, Dn – qui ne sont **pas nécessairement tous distincts** - est constituée de deux parties, un en-tête (schéma de la relation) et un corps (ensemble de lignes).
L'en-tête est un ensemble fixé d'attributs, des couples < Nom – Attribut; Nom – Domaine>, {< A1 : D1>, < A2 : D2>,…, < An : Dn >}, tel que chaque attribut Aj correspond exactement à un des domaines sous-jacents Dj (j = 1, 2, …, n). Les noms des attributs A1, A2 , …, An sont tous distincts.

- Le corps consiste en un sous-ensemble de **n-uplets.**
- Le nombre de n-uplets correspond à la **cardinalité** de la relation, alors que le nombre d'attributs correspond au **degré** de la relation.

Chaque relation possède une interprétation ou une signification voulue.

La relation fournisseur signifie : le fournisseur ayant un numéro de fournisseur *f* possède un nom *n*, un code *c*, et se trouve dans une ville *v*. De plus, deux fournisseurs n'ont jamais le même numéro.

Formellement, la déclaration précédente, est un exemple de ce qu'on appelle **Prédicat,** ou fonction ayant une valeur de vérité. Ce prédicat (cette fonction) comprend quatre (04) arguments. Substituer les arguments par des valeurs équivaut à invoquer la fonction ou à ″instancier″ le prédicat et de ce fait, à produire une expression, appelée **proposition** qui peut être vraie ou fausse.

Par exemple, la substitution :

{< NF = f 1>, <NOM= Haroun>, < CODE = 120>, < VILLE = Alger >}

produit une proposition vraie. Par contre, la substitution

{< NF = f1 >, < NOM = Bouzid >, < CODE = 120>, < VILLE = Alger >} ,

produit une proposition fausse.

A tout instant, la relation contient exactement les n-uplets qui font que le prédicat va être évalué à ″Vrai″ à cet instant là.

Une relation correspond à ce que nous avons appelé jusqu'ici une table. Un n-uplet correspond à une ligne d'une table, un attribut à une colonne.