Chapitre 2
Modèle relationnel

1. Introduction au modèle relationnel

Le modèle relationnel de données a été défini en 1970 par le chercheur Britannique Edgar Frank Codd. L'objectif était de surmonter les lacunes des modèles hiérarchique et réseaux qui ne sont pas capables de gérer les BD volumineuses, ne garantissent pas l'intégrité de données, ne traitent pas les redondances de données, etc. Les Systèmes de Gestion de Base de données Relationnelles (SGBD) commerciaux sont apparus dans les années 80 avec des outils tel qu'ORACLE.

Le modèle relationnel est un modèle facile, simple et qui repose sur des bases théoriques solides : la théorie des ensembles et la logique des prédicats du premier ordre. Ce modèle permet : l'amélioration de l'indépendance logique et physique ainsi que l'amélioration de l'intégrité et de la confidentialité, la proposition de schémas de données faciles à utiliser, l'optimisation de l'accès à la BD, etc.

1.1. Définition du modèle relationnel

Dans ce modèle, un concept unique représente les objets et les associations, c'est la *relation*. Les relations sont des tableaux à deux dimensions appelées *Tables*. Dans une table, une ligne correspond à un enregistrement et une colonne à un champ de cet enregistrement. Des opérateurs de l'algèbre relationnel permettent la manipulation des données et l'ensemble de contraintes d'intégrité définissent l'état cohérent de la base.

Exemple :

Enseignant

NEns	Nom	Prénom	Département
20	Nasri	Ahmed	Marketing
40	Taleb	Mohammed	Informatique
200	Hadad	Omar	Comptabilité

Le nom de la relation de l'exemple ci-dessus est Enseignant, cette relation décrit les enseignants tel que chaque ligne de la table correspond à une occurrence.

Par exemple : <40, Taleb, Mohammed, Informatique>.

1.2. Concepts de base (attribut, Tuple, domaine, relation)

- **Attribut :** Le nom donné à une colonne d'une relation est appelé *attribut*. L'attribut prend sa valeur dans un domaine. Par exemple **:** NEns et Nom sont des attributs de la relation Enseignant.

- **Tuples** : Les lignes d'une table sont appelées n-uplet ou *tuples* en anglais.

- **Domaine** : Un *domaine* est un ensemble de valeurs que peut prendre un attribut. Cet ensemble de valeurs peut être finies ou infinies (e.g., chaine de caractère, entier, etc.).
  Par exemple : Le domaine des couleurs primaires : DC= {rouge, bleu, jaune}.
  Le domaine des booléens : DB= {0,1}.

- **Relation** : Une *relation* est un ensemble de tuples déterminé par un nom et qui peut être exprimée en intension ou en extension. En d'autres termes, une relation est un sous-ensemble du produit cartésien d'une liste de domaines caractérisé par un nom.
  Exemple de relation :

Client

N^0Cli	Nom	Prénom	Datnais
101	Talbi	Ahmed	02/03/2000
102	Dali	Mohammed	05/06/1999

1.3. Schéma de relation

Le schéma de la relation est présenté par le nom de la relation suivi par les attributs et leurs domaines d'application (i.e. les tuples de la relation ne sont pas listées). La clé de la relation est constituée d'un sous-ensemble des attributs de la relation. Ce sous-ensemble sera souligné dans le schéma.

Exemple **:** CLIENT (N^0Cli : entier, Nom : CC, Prénom : CC, Datnais : Date).

Souvent, le schéma de relation se limite au nom de la relation suivi de ses attributs pour des raisons d'allégement d'écriture.

Exemple : CLIENT (N^0Cli, Nom, Prénom, Datnais).

Notons que le schéma d'une relation représente son **intention** (i.e. les propriétés communes et invariantes des tuples qu'elle va contenir).

Les tuples de la relation présentée en extension sont visibles (i.e. listés). La relation dans ce cas est présentée sous forme de tableau. Notons que :

- Le *degré* de la relation indique son nombre d'attributs (e.g., le degré de la relation Client est 4).
- Le nombre de tuples d'une relation représente la *cardinalité* de cette relation (e.g., la cardinalité de la relation Client est 2).

1.4. Passage du modèle Entité/ Association au modèle relationnel

Le passage du modèle E/A vers le relationnel est possible via des règles de transformation.

Règle 1 : Toute entité devient une relation dans laquelle l'identifiant de l'entité devient clé primaire de la relation et les attributs de l'entité deviennent attributs de la relation (voir l'exemple dans la figure 2.1).

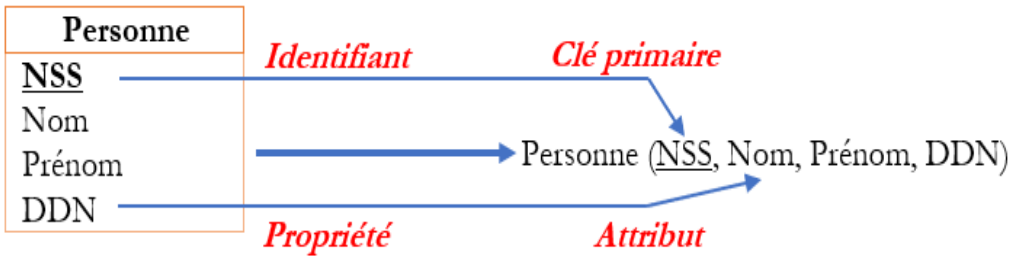

Figure 2.1. Exemple d'application de la règle1 de transformation.

Règle 2 : Une association binaire de cardinalité 1,1 devient une relation portant dans la relation fille la clé primaire de la relation mère et l'attribut ajouté s'appelle clé étrangère dont le symbole est # (voir l'exemple dans la figure 2.2).

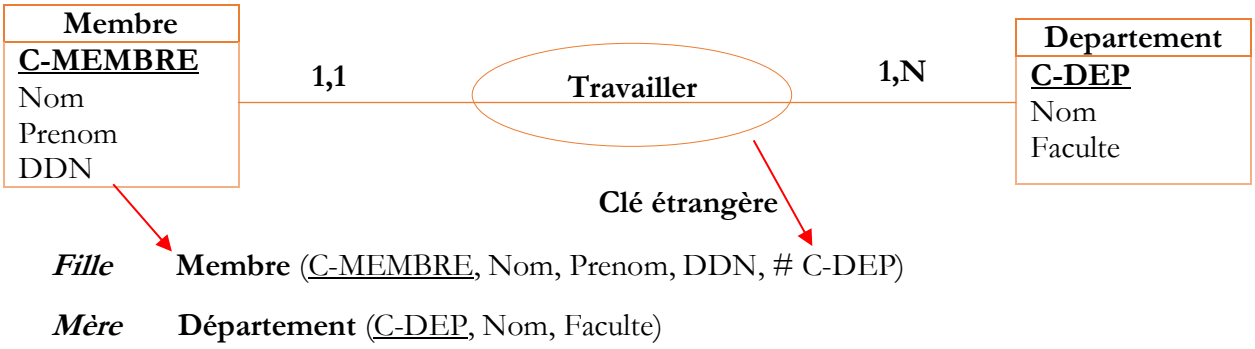

Figure 2.2. Exemple d'application de la règle 2.

Règle 3 : Dans une association binaire de cardinalité plusieurs à plusieurs, chaque entité devient une relation et une relation particulière est créée et aura comme clés étrangères les identifiants des 2 entités associées en ajoutant la ou les propriétés éventuelles de cette relation.

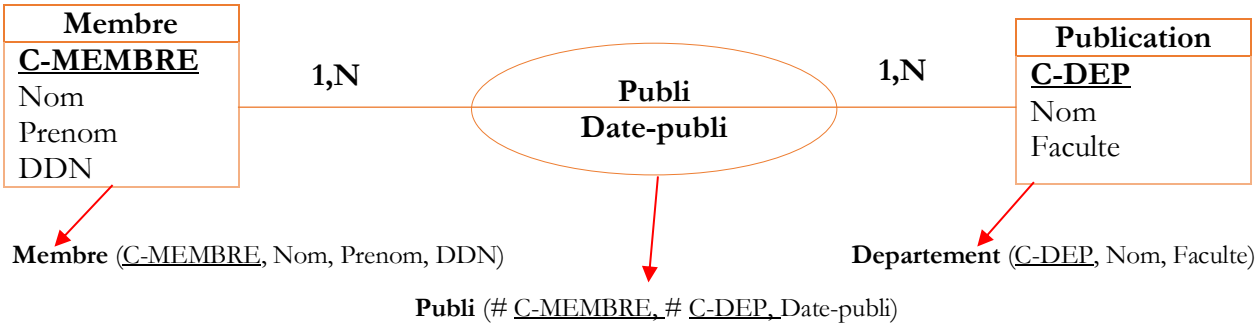

Figure 2.3. Exemple d'application de la règle 3.

Règle 4 : Une association de dimension supérieur à 2, s'écrit suivant la règle 3.

2. Normalisation

La théorie de la normalisation consiste en la décomposition des informations en plusieurs relations en se basant sur les dépendances fonctionnelles. Les buts de la normalisation sont :

- Limitation des redondances des tuples, ce qui permet la réduction de l'espace de stockage.
- Suppression des valeurs NULL et par conséquent, suppression des difficultés sur les jointures et les fonctions d'agrégation.
- Pour limiter les incohérences au sein des données.
- Limitation des problèmes de mise à jour et amélioration des performances des traitements.

2.1. Dépendances fonctionnelles (DFs)

Les dépendances fonctionnelles (Functional dependency) permettent l'établissement des liens entre attributs ou groupe d'attributs.

2.1.1. Définition

Soit R (A1, A2… An) avec n ≥2 un schéma de relation, soit X et Y deux sous-ensembles d'attribut de {A1, A2… An} tel que X∩Y=A. On dit que X → Y (X détermine Y, ou Y dépend fonctionnellement de X) si : ∀ les tuples (x,y) et (x',y') de R, x=x' → y=y'.

Par exemple : Type → Marque.

Cela veut dire qu'à chaque valeur de X dans R correspond une seule valeur de Y. Les DFs déterminent les contraintes entre les attributs (e.g., deux individus peuvent avoir le même nom et le même prénom mais jamais le même numéro). Les dépendances fonctionnelles sont spécifiées théoriquement par l'administrateur puis contrôlées par les SGBD.

2.1.2. Propriétés des DFs

Les trois règles suivantes sont connues sous le nom d'axiomes d'Armstrong et permettent d'effectuer des inférences de DFs à partir d'autre DFs :

- *Réflexivité* : pour tout Y⊆ X ⇒ X → Y : tout ensemble d'attributs détermine lui-même ou une partie de lui. Par exemple :
  NE, NP → NE, NP
  NE, NP → NE

- *Augmentation* : si X → Y ⇒ X, Z → Y, Z ; si X détermine Y, alors les deux ensembles d'attributs peuvent être enrichis par un même troisième.
  Exemples :

  1) Jour, Heure, sigleCours, noGroupe → Jour, Heure, codeProfesseur

     Augmentation de : SigleCours, noGroupe → codeProfesseur

  2) Titre_film→ Genre_Film

Par augmentation : Titre_Film, Année→ Genre_Film, Année

- *Transitivité* : si X → Y et Y → Z ⇒ X → Z. Par exemple :

  SigleCours, noGroupe → codeProfesseur et codeProfesseur → Local

  Par transitivité : sigleCours, noGroupe → Local

Plusieurs autres règles peuvent être déduites de ces axiomes de base :

- *Union* : si X → Y et X → Z ⇒ X → Y,Z.
  Exemple : NumEtud → Nom, Prenom et NumEtud → AnneeEtud
  Alors NumEtud → Nom, Prenom, AnneeEtud

- *Pseudo-transitivité* : si X → Y et WY → Z ⇒ WX → Z.
  Exemple : NumEns → Grade et Grade, NomEns → Salaire
  Alors NumEns, NomEns → Salaire

- *Décomposition* : si X → Y et Z ⊆ Y alors X → Z.
  Exemple : Titre_Film → Genre_Film, Année
  Par décomposition : Titre_Film → Genre_Film

2.1.3. Types des DFs

- **DF triviale :** X → Y est triviale si Y ⊆ X.

- **DF élémentaire :** X→ Y est élémentaire si ∀ X'⊆ X, X' ne détermine pas Y (i.e. Y ne dépend pas d'une partie de X).
  Exemple : NumProduit → NomProduit est une DF élémentaire.
  NumFacture, NumProduit → NomProduit n'est pas élémentaire.

- **DF canonique** : une DF X→ Y est dite canonique si ca partie droite est réduite à un seul attribut. Toute DFs qui n'est pas canonique peut être transformées par décomposition en DFs canoniques. Par exemple : NumCli→NomCL, PrénomCl, AgeCl n'est pas canonique, par décomposition, elle donne trois DFs canoniques :
  NumCli →NomCL
  NumCli →PrénomCl
  NumCli →AgeCl

- **DF directe :** une DF X→ Y est dite directe si elle est élémentaire et Y ne dépend pas par transitivité de X. Autrement dit, la dépendance entre X et Y ne peut pas être obtenue par transitivité (∄ Z/ X→Z et Z →Y).

2.1.4. Graphe de Dépendances Fonctionnelles (GDF)

Le graphe des DFs permet la visualisation facile des DFs et d'isoler les DFs élémentaires. Les sommets sont les attributs et les arcs sont les DFs.

Exemple :

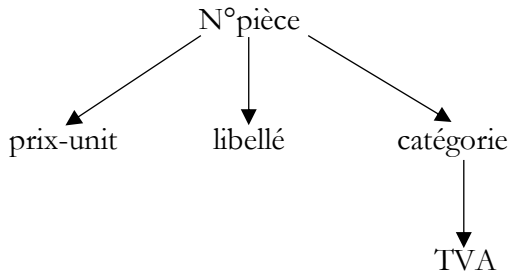

Figure 2.4. Graphe de dépendance fonctionnelle.

- Pièce (N°pièce, prix-unit, libellé, catégorie).
- Catégorie (catégorie, TVA).

2.2. Fermeture transitive (Transitive closure)

La fermeture transitive d'un ensemble F de DF est un ensemble notée F+ composé de F et de l'ensemble des DFs déduits par transitivité.

Exemple : à partir de l'ensemble de DF :

F = {NV → TYPE ; TYPE → MARQUE ; TYPE → PUISSANCE}

On déduit la fermeture transitive : F+ = F ∪ {NV → MARQUE ; NV → PUISSANCE}.

Remarque : Si deux ensembles de DFs ont la même fermeture transitive, alors ils sont équivalents.

Par suite, il est intéressant de déterminer un sous-ensemble minimal de DFs permettant de générer tout les autres. C'est la *couverture minimale* d'un ensemble de DFs [1].

2.3. Couverture minimale (Minimal cover)

La couverture minimale appelée aussi couverture irredondante et notée IRR(F) est un ensemble F de DFEs (élémentaires) associé à un ensemble d'attributs vérifiant les propriétés suivantes :

- Aucune dépendance dans F n'est redondante. Cela signifie que pour toute DF *f* de F, F – {*f*} n'est pas équivalent à F.
- Toute DFE des attributs est dans la fermeture transitive de F.

La couverture minimale est un sous-ensemble minimal de DFEs permettant de générer toutes les autres.

Algorithme de calcul de la couverture minimale IRR(F) [5] :

Le rôle de cet algorithme est de détecter les DFs redondantes (i.e. les DFs qu'on peut déduire à partir des autres DFs restantes en utilisant les axiome d'Amstrong).

Couverture(F);
Begin
IRR := F
Remplacer Chaque DF X→(A1,...,An) par n DF X→A1,..., X→An
Pour chaque DF : *f* =X → A dans IRR
SI (IRR –{*f*}) Implique {*f*} **Alors** IRR = IRR -{*f*}
FPOUR
Return IRR
Fin

Algorithme détaillé de IRR(F) :

Cet algorithme détaille la notion de redondance de DFs. On dit donc qu'une Df *f* est redondante si la fermeture de sa partie gauche sur l'ensemble des DFs restantes (F-*f*) comporte la partie droite de *f*. Cela signifie que pour une DF X→Y et même en enlevant *f*, le lien sémantique entre X et Y est conservé.

Procédure Couverture Minimale(F:Ensemble de DF)
Var CM : Ensemble de DF
Début
CM = F
Décomposer les parties droites de DF
Pour chaque DF f : X→Y MC snad **Faire**
Début
Calculer X+(F-{*f*})
Si Y est inclus dans X+ **Alors** CM=CM-{*f*}
FPour
Retourner CM
Fin

La couverture minimale va constituer un élément essentiel pour composer des relations sans perte d'informations directement à partir des attributs.

2.4. Clé de relation

Dans le modèle relationnel, la clé de relation est un concept de base. En effet, La clé d'une relation R est un ensemble minimal d'attributs de R permettant de déterminer tous les attributs de R. Une définition plus formelle de la clé d'une relation est comme suit :

Soit R une relation ayant l'ensemble des attributs A et soit X un sous-ensemble de A. X est une clé de R si ∀ a∈ A, X→ a.

- **Clé minimale :** Un sous-ensemble d'attributs X d'une relation R est dit clé minimale si :
  - X est une clé ;

- Toute dépendance fonctionnelle X→ a de R est élémentaire.

- **Clé candidate/ primaire** : Une relation R peut avoir plusieurs clés appelées *clés candidates*. Parmi les clés candidates une est choisi comme en général comme clé primaire.

- **Superclé :** Tout ensemble d'attributs incluant tous les attributs de la clé primaire est dit *superclé*. Notons que l'ensemble de tous les attributs d'une relation est une superclé. En outre, la clé primaire est une superclé.

- **Clé étrangère :** Afin de permettre au SGBDR de maintenir la cohérence des lignes de deux relations, la clé étrangère sert de lien entre deux relations d'une même BDD.

2.5. Contrainte d'intégrité

Lors des mises à jour des données d'une BD, le SGBD doit assurer la cohérence des données. Ces données ne sont pas indépendantes, mais obéissent à des règles sémantiques appelées **contraintes d'intégrité** [1].

Il existe différents types de contraintes d'intégrité :

- *Unicité de clé* : une relation doit posséder une clé primaire dont la valeur est unique. C'est un groupe d'attributs non nul dont la valeur permet de déterminer un tuple unique dans une table

- *Contrainte Référentielle :* représente une association entre deux tables dites de clé étrangère imposant que la valeur d'attribut de la relation R1 apparaît comme valeur de clé dans une autre relation R2.

- *Contrainte de domaine* : permet de restreindre les valeurs d'un domaine (e.g., une note doit être comprise entre 0 et 20).

- *Contrainte de non nullité* : la valeur d'un attribut doit être renseignée. Un attribut d'une clé ne doit pas posséder de valeurs nulles (vides).

Exemple :

- Le nombre d'exemplaires de chaque OUVRAGE doit être supérieur à 0 (zéro).
- Chaque OUVRAGE doit avoir au moins un auteur.

2.6. Formes normales

La normalisation sert à concevoir d'une manière correcte le schéma d'une base de données relationnelle. Les trois premières formes normales ont pour objectif de permettre la décomposition de relations sans perdre d'informations, à partir de la notion de dépendance fonctionnelle [6]. Chaque forme normale est une progression vers des relations présentant moins de redondances.

Cette décomposition permet d'éliminer la redondance des données ainsi que les anomalies de mise à jour.

Exemples :

Soit la relation universelle (i.e. relation qui regroupe toutes les informations à stocker) suivante : Etudiant (NEtud, Nom, Prénom, Matière, Note)

NEtud	**Nom**	**Prénom**	**Matière**	**Note**
101	Nasri	Ahmed	Bases de données	10
101	Nasri	Ahmed	Réseaux	13
101	Nasri	Ahmed	Génie Logiciel	08
102	Haddad	Omar	Bases de données	09
102	Haddad	Omar	Réseaux	11
102	Haddad	Omar	Génie Logiciel	07

On remarque que le numéro de l'étudiant, son nom et son prénom, sont **répétés** autant de fois que le nombre des matières ⇒ **redondance** des données.

Soit la relation : Livraison (N°fourn, adrF, N°prod, prixP, qté)

N°fourn	adrF	N°prod	prixP	qté
3	Annaba	52	65	100
22	Alger	10	15	50
22	Alger	33	10	120
3	Annaba	33	10	50
3	Constantine	10	15	200

- Si un fournisseur change d'adresse et qu'un seul tuple est mis à jour, ou si un nouveau tuple est inséré pour un fournisseur connu avec une adresse différente ⇒ **incohérence.**
- Si un fournisseur n'a pas de livraison en cours ⇒ perte d'adresse.

La normalisation est un processus qui permet d'éviter la *redondance, la perte* et *l'incohérence* de données dans le modèle relationnel.

2.6.1. Première Forme Normale (First normal form) 1FN

Une relation est en première forme normale si :

- Elle possède une clé,
- Tout attribut contient une valeur atomique (i.e. les attributs ne sont pas multivalués).

Notes : Un attribut est multivalué n'est qu'une relation dans une relation. Un attribut atomique n'a à un instant donné qu'une seule valeur.

La 1FN consiste à éviter les domaines composés de plusieurs valeurs.

Exemple : Soit la relation Etudiant (NEtud, Matière, Nom, Prénom, Note).

Dans le cas où on prend en considération les notes de rattrapage, l'attribut Note aura deux valeurs. Donc, la relation n'est pas en 1FN.

Afin que la relation Etudiant soit en 1FN, en décompose Note en : NoteExamen et NoteRattr.

On aura la relation : Etudiant (NEtud, Matière, Nom, Prénom, NoteExamen, NoteRattr).

2.6.2. Deuxième Forme Normale (Second normal form) 2FN

Une relation R est en deuxième forme normale si et seulement si :

- Elle est en 1FN.
- Toutes les DFs sont élémentaire (les attributs qui n'appartiennent pas à une clé ne dépendent pas d'une partie de cette clé).

Afin de normaliser une relation en 2FN, il faut décomposer la relation R en plusieurs relations. La décomposition se fait en plaçant les attributs qui violent la 2FN dans une nouvelle relation avec la clé primaire ou la clé partielle.

Exemple : Soi la relation : Livraison (N°fourn, N°prod, prixP, qté).

Avec : N°prod → prixP

Cette relation n'est pas en 2FN car N°prod → prixP (partie de clé détermine un attribut non clé).

La relation sera donc décomposée en :

- Livraison (N°fourn, N°prod, qté)
- Produit (N°prod, prixP)

Notes :

- Une relation en 2FN traduit le fait que les attributs non clé dépendent complétement de la clé.
- Une relation en 2FN peut contenir des informations redondantes. Donc, la 2FN ne permet pas d'éviter toutes les redondances.

2.6.3. Troisième Forme Normale (Third nomal form) 3FN

Une relation R est en troisième forme normale si et seulement si :

- Elle est en 2FN.
- Tout attribut n'appartenant pas à la clé ne dépend pas d'un autre attribut non clé. Cela veut dire que toutes les dépendances fonctionnelles sont directes.

Exemple 1 :

Soit : Enseignant (NumEns, Non, Prénom, Grade, ChargeHoraire).

Avec : Grade → ChargeHoraire. La relation n'est pas en 3FN car un attribut non clé détermine un attribut non clé.

Cette relation doit être scindée en deux relations :

Enseignant (NumEns, Non, Prénom, Grade)

Charge (Grade, ChargeHoraire).

Exemple 2 :

Soit : R(Vol, AéroportDépart, Appareil, AéroportArrivée).

Avec : Vol, AéroportDépart → AéroportArrivée et Vol →Appareil.

R n'est pas en 2FN car une partie de la clé détermine un attribut non clé. La décomposition de R donne : R1(Vol →Appareil) et R2(Vol, AéroportDépart, AéroportArrivée).

2.6.4. Forme Normale FN de Boyce-Codd BCNF (Boyce-Codd normal form)

La 3FN est insuffisante lorsqu'une relation admet plusieurs clés candidates, car des anomalies peuvent avoir lieu. Pour cela, Boyce et Codd ont introduit la forme normale BCNF.

Une relation est en BCNF si et seulement si :

- Elle est en 3FN.
- Pour toute dépendances fonctionnelles, la partie gauche est une clé.

Exemple : Soit la relation R(A,B,C)

Avec : A,B →C et C→ A

Cette relation n'est pas en BCNF car le membre gauche de la DF C→ A n'est pas une clé. On décompose la relation en : R1(B,C) et R2(A, C).

Cette décomposition ne préserve pas la DF A,B → C.

Notes :

- Il n'est pas toujours possible d'aboutir à une décomposition en BCNF sans perte d'information et qui préserve les DFs.
- On mesure la qualité d'une relation par son degré de normalisation.

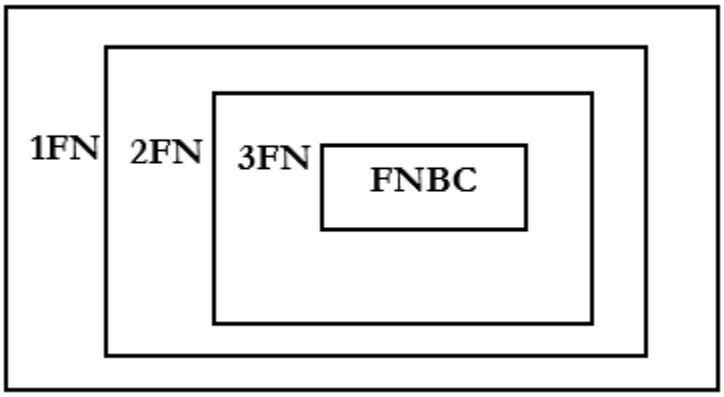

Figure 2.5. Formes normales.

2.7. Schéma de base de données

Le schéma d'une base de données est défini par l'ensemble des schémas des relations qui composent la base de données. Le schéma de la BDR dit comment les données sont organisées dans la base [9].

Il indique quelles tables ou relations constituent la base de données, ainsi que les champs inclus dans chaque table. Un schéma contient un groupe de tables, alors qu'une BD contient un groupe de schémas (voir l'exemple dans la figure 2.4).

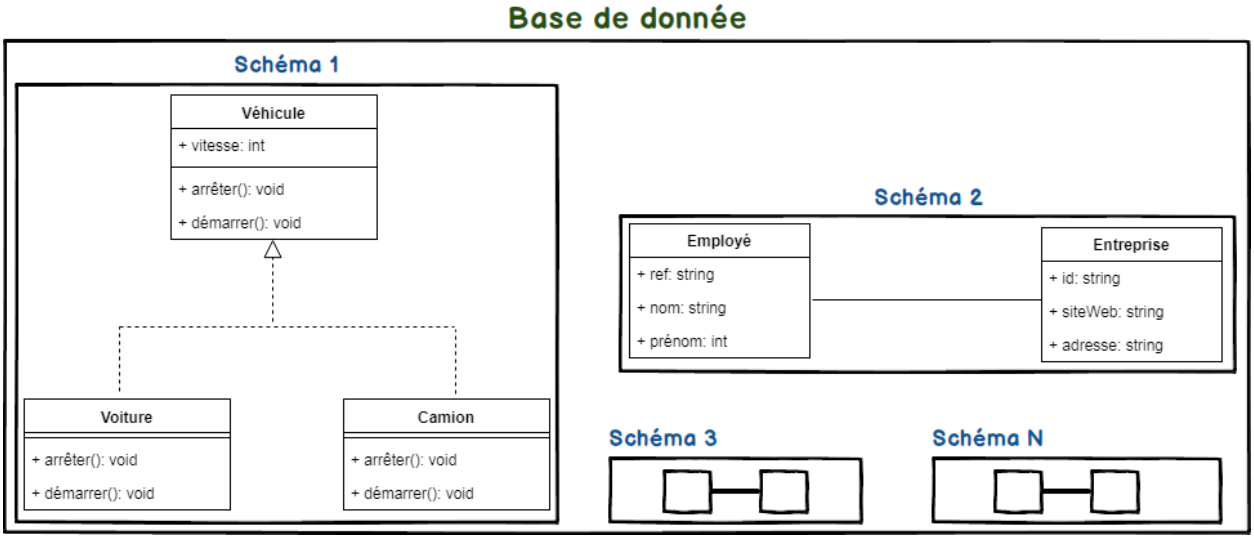

Figure 2.4. Différence entre Schéma de base de données et Base de données [12].

3. Modèle relationnel logique (SQL)

3.1. Table, colonne et ligne

- RELATION = **TABLE**

A1	B1	C1	D1
A2	B2	C2	D2
A3	B3	C3	D3
A4	B4	C4	D4

- ÉLÉMENT ou n-uplet = **LIGNE**

LIGNE → 1 élément

A1	B1	C1	D1

On ne peut pas avoir 2 lignes identiques.

- ATTRIBUT = **COLONNE**

A1			
A2			
A3			

↑ COLONNE

Attribut ou Propriété

3.2. Description de SQL (Structured Query Langage)

SQL (Structured Query Langage ou bien langage de requête structuré) est un langage informatique standard. Il permet la communication avec les SGBDRs et a été défini par l'ANSI (American National Standard Institute) et l'ISO (International Standards Organization).

Le succès du langage SQL est dû essentiellement à sa simplicité et au fait qu'il s'appuie sur le schéma conceptuel pour énoncer des requêtes en laissant le SGBD responsable de la stratégie d'exécution [3].

SQL est un langage déclaratif qui permet d'interroger une base de données sans se soucier de la représentation interne (physique) des données, de leur localisation, des chemins d'accès ou des algorithmes nécessaires [4].

Néanmoins, le langage SQL ne possède pas la puissance d'un langage de programmation : entrées/sorties, instructions conditionnelles, boucles et affectations. Pour certains traitements il est donc nécessaire de coupler le langage SQL avec un langage de programmation plus complet [3].

SQL est un langage relationnel, c'est-à-dire, il manipule des tables via des requêtes et produisent des tables.

Des versions différentes ont été proposées :

- SQL 1 86 : la base.
- SQL 1 89 : l'intégrité.
- SQL 2 92 : la nouvelle norme.
- SQL 3 99 : les évolutions objets.

SQL se compose de cinq parties (voir Tableau 2.1).

TCL Set Transaction, Commit, rollback		
DDL Create Alter Drop	**DML** Insert Update Delete Select	**DCL** Connect Grant Revoke
Programmation SQL Declare, Fetch, Prepare, Describe, Execute		

Tableau 2.1. Composants du langage SQL [5].

- Data Definition Language (Langage de Définition des Données) est la partie du SQL permettant la création des bases de données, etc.
- Data Manipulation Language (Langage de Manipulation des Données) est la partie de traitement des données (ajout, suppression, etc.).
- Data Control Language (Langage de contrôle des Données).
- TCL pour gestion des transactions.
- La programmation SQL dynamique.

3.3. Définition de données

3.3.1. Création de table (CREATE)

La commande **CREATE TABLE** permet de créer les tables SQL comme suit :

CREATE TABLE <nom de table> (<élément de table>$^{+}$)

Un élément de table est soit une définition de colonne ou de contrainte :

<ÉLÉMENT DE TABLE> ::= <DÉFINITION DE COLONNE> |

<CONTRAINTE DE TABLE>

La syntaxe de la définition de colonne est comme suit :

<DÉFINITION DE COLONNE> : := <NOM DE COLONNE> <TYPE DE DONNÉES>

[<CLAUSE DÉFAUT>] [<CONTRAINTE DE COLONNE>]

3.3.1.1. Contrainte de colonne

La contrainte de colonne peut être :

- Interdictions des valeurs nulles (existence des valeurs) des attributs en utilisant le mot clé **NOT NULL**.
- Utilisation de la clause **UNIQUE** afin de s'assurer de l'unicité des valeurs des attributs (pas de valeurs dupliquées dans les colonnes).
- **CHECK** (condition) pour vérifier que l'attribut réalise la condition lors de l'insertion de n-uplets.
- **PRIMARY KEY :** Clé primaire.
- **DEFAULT value** pour spécifier la valeur par défaut de l'attribut.

3.3.1.2. Contrainte table

La contrainte de table peut être :

FOREIGN KEY (colonne…) REFERENCES table [(colonne…)]

Soit les deux tables :

Personne

PersonID	NomP	PrénomP
1	Talbi	Ali
2	Bendali	Ahmed
3	Azizi	Farid

Ordre

OrdreID	NumeroOrdre	PersonID
1	77895	3
2	44678	3
3	22456	2
4	24562	1

L'attribut "PersonID" est commun entre les deux tables et il est clé primaire dans la table "Personne". L'attribut "PersonID" est une clé étrangère dans la table "Ordre"

La requête SQL suivante créer une clé étrangère dans la colonne "PersonID" lorsque la table "Ordre" est créé :

```
CREATE TABLE Ordre (
   OrdreID int NOT NULL,
   NumeroOrdre int NOT NULL,
   PersonID int,
   PRIMARY KEY (OrdreID),
   FOREIGN KEY (PersonID) REFERENCES Personne(PersonID)
);
```

Les types de données utilisés lors de la création des tables en SQL dans ACCESS sont comme suit :

- Booléen : BIT
- Nombre entier : SHORT(entier), SMALLINT(entier), LONG(enter long), INTEGER(entier long).
- Nombre réel : SINGLE(réel simple), DOUBLE(réel double), Numeric(réel double).
- Monétaire : CURRENCY, MONEY.
- Date/heure : DATE, TIME, DATETIME.
- Texte : VARCHAR(255) : taille variante, CHAR(n) ou Text(n) : n est le nombre de caractères.

Exemple 1 :

```
CREATE TABLE AVION(
  Num_avion   SMALLINT              PRIMARY KEY,
  TYPE   VARCHAR(10)                NOT NULL,
  CONSTRUCTEUR VARCHAR(20) NOT NULL,
  CAPACITE SMALLINT                 CHECK (CAPACITE >0),
  COMPAGNIE CHAR(35)                NOT NULL );
```

Exemple 2 :

```
Create table realisateur (
   id_real integer primary key,
   nom varchar(16) not null,
   prenom varchar(16) not null);
```

3.3.2. Modification de schéma (ALTER, DROP)

3.3.2.1. Suppression d'une table

Les tables SQL sont supprimées en utilisant la syntaxe suivante :

DROP TABLE nom_table ;

Exemple : **DROP TABLE** employé ;

La suppression d'une table permet d'éliminer :

- La structure de cette table.
- Toutes les données qu'elle contient.
- Les indexes associés.

Notes :

- Une clé étrangère d'une autre table ne doit pas référencer la table à supprimée.
- Les clés étrangères qui référencent la table à supprimée peuvent être supprimées en Oracle en ajoutant le mot clé CASCADE à la fin.

3.3.2.2. Modification d'une table

L'instruction **ALTER TABLE** permet de modifier la structure des tables SQL en ajoutant ou en modifiant la structure d'une table. La syntaxe est comme suit :

ALTER TABLE nom-table modification ;

Plusieurs situations peuvent avoir lieu comme suit :

Syntaxe	Explication
ALTER TABLE *nom_table* ADD *att type* NOT NULL ;	Insérer à la table un attribut *att* qui contient des données correspondant à *type*. Exemple : ALTER TABLE utilisateur ADD adresse_rue VARCHAR(255) ;
ALTER TABLE *nom_table* RENAME COLUMN *att* TO *nouvel att*;	Changer le nom d'un attribut en un nouveau nom.
ALTER TABLE *nom_table* DROP COLUMN att ;	Supprimer un attribut de la table. Exemple : ALTER TABLE Etudiant DROP COLUMN Date-naiss ;
ALTER TABLE *nom_table* ADD CONSTRAINT *nom contrainte* ;	Ajouter une contrainte dans la table.
ALTER TABLE *nom_table* DROP PRIMARY KEY ;	Supprimer la clé primaire de la table.
ALTER TABLE *nom_table* DROP FOREIGN KEY *nom clé* ;	Supprimer la clé étrangère.

Tableau 2.2. Situations possibles de la commande ALTER TABLE.

L'instruction suivante permet le renommage d'une table :

RENAME ancien_nom TO nouveau_nom;

3.4. Manipulation des données sur les tables (INSERT, DELETE, UPDATE)

Le Langage de Manipulation des Données (Data Manipulation Language) est la partie du SQL qui permet la manipulation des *lignes* des tables à travers trois instructions : INSERT INTO, UPDATE et DELETE.

3.4.1. INSERT INTO

La commande **INSERT INTO** permet l'insertion de lignes dans une table. La syntaxe de la commande d'insertion est comme suit :

INSERT INTO nom_table (nom_colonne_1, nom_colonne_2, ...) **VALUES** ('valeur 1', 'valeur 2', ...)

Dans le cas où la liste des noms de colonnes est omise (n'est pas spécifiée), tous les attributs de la relation doivent être fournis dans l'ordre de déclaration. Si uniquement quelques colonnes sont spécifiées, les autres sont insérées avec la valeur NULL.

Exemples :

- Insertion d'une seule ligne aves spécification des colonnes.

INSERT INTO client (prenom, nom, ville, age) **VALUES** ('Talbi', 'Ahmed', 'Boumerdes', 20)

- Insertion d'une seule ligne sans spécification des colonnes.

INSERT INTO client **VALUES** ('Talbi', 'Ahmed', 'Boumerdes', 20)

- Insertion de plusieurs lignes.

INSERT INTO client (prenom, nom, ville, age)

VALUES

('Talbi', 'Ahmed', 'Boumerdes', 20),

('Mansouri', 'Ridha', 'Oran', 24),

('Salem', 'Omar', 'Alger', 29)

3.4.2. UPDATE

La commande **UPDATE** permet la mise à jour (Modification des lignes) des valeurs d'attributs de tuples existants. La modification peut concerner tous les tuples de la table en fournissant des valeurs à changées, ou en élaborant des valeurs via une expression. On peut définir des conditions de mise à jour dans une clause WHERE qui est une clause optionnelle. La syntaxe de la commande UPDATE est la suivante :

UPDATE nom_table
SET nom_col_1 = {expression_1 | (SELECT ...) },
nom_col_2 = {expression_2 | (SELECT ...) },
...
nom_col_n = {expression_n | (SELECT ...) }
WHERE predicat

Exemple 1 :

Modifier l'email de l'auteur Ahmed Talbi.

UPDATE Auteur

SET Mail= 'Ahmed.Talbi@gmail.com'

WHERE Nom='Talbi' **AND** Prénom='Ahmed'

Exemple 2 :

Ajouter deux points aux étudiants qui ont une note inférieure à 8.

UPDATE Etudiant

SET note=note+2

WHERE note<8

3.4.3. DELETE

La commande **DELETE** permet d'enlever (supprimer) d'une relation des tuples existants. La syntaxe de la commande est la suivante :

DELETE FROM nom_table **WHERE** condition

Tout les tuples pour lesquelles le prédicat est évalué à vrai sont supprimés. Par contre si la clause WHERE ne figure pas dans la commande, tous les tuples de la relation sont supprimés.

Exemple 1 :

Supprimer toutes les lignes de la table Etudiant.

DELETE FROM Etudiant

Exemple 2 :

Supprimer les étudiants dont la Moyenne est inférieure à 10.

DELETE FROM Etudiant **WHERE** Moy_etud< 10